Ariel

Rapid #: -4543785

8

IP: 150.135.238.50

Status	Rapid Code	Branch Name	Start Date
Pending	GZN	Main Library	6/28/2011 5:18:34 PM

CALL #: **QA76 .T726x**
LOCATION: **GZN :: Main Library :: stacks**
TYPE: Article CC:CCL
JOURNAL TITLE: Transactions of the Society for Computer Simulation
USER JOURNAL TITLE: Transactions of the Society for Computer Simulation
GZN CATALOG TITLE: Transactions of the Society for Computer Simulation.
ARTICLE TITLE: Knowledge-Based Simulation Design Methodology: A Flexible Test Architecture Application
ARTICLE AUTHOR:
VOLUME:
ISSUE:
MONTH:
YEAR: 1990
PAGES:
ISSN: 0740-6797
OCLC #: GZN OCLC #: 9991936
CROSS REFERENCE ID: [TN:957018][ODYSSEY:150.135.238.6/ILL]
VERIFIED:

BORROWER: **AZU :: Main Library**
PATRON: **George Hwang**
PATRON ID: hwangg
PATRON ADDRESS:
PATRON PHONE:
PATRON FAX:
PATRON E-MAIL:
PATRON DEPT:
PATRON STATUS:
PATRON NOTES:

RAPID ILL

System Date/Time: 6/29/2011 7:31:09 AM MST

the Causality Relation" by
deling technique, based on
oundation for a simulation
such as "Will event A ever
nberg use histories (stored
nected with the simulation.

Key words: *Knowledge-Based System Design, Simulation, Testing Methodology, Hierarchical, Modular Systems.*

Knowledge-Based Simulation Design Methodology: A Flexible Test Architecture Application*

Jerzy W. Rozenblit
Bernard P. Zeigler

AI Simulation Group
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721

ABSTRACT

This article presents Knowledge-Based Simulation Design Methodology and its application in the domain of flexible testing. Basic concepts and techniques of the methodology are introduced and employed to design a flexible Printed Circuit Board (PCB) test architecture. A PCB testing framework is proposed. The framework involves the following phases: A board under test is represented using the *entity structure* formalism. This representation, augmented with test criteria, is used to select a test resolution (degree of test detail). The fault modeling process is supported by a class of tree models specified in Discrete Event System Specification (DEVS). Tree models with fault locations represent PCBs for which a test station architecture and testing strategy are being designed. The test strategy selection and the configuration of the test station are generated from the generic entity structure representation of a tester. This generation process uses a production rule-based approach. The tester architecture is modeled and simulated using DEVS-Scheme simulation language. Alternative tester architectures and test strategies can be compared through a simulation modeling study prior to an actual set-up of a test cell. This approach is expected to result in improvements in test/rework time, first pass yield, and reduced scrap and work-in-process inventory.

1. INTRODUCTION

This article presents a knowledge-based simulation design (KBSD) framework combining Artificial Intelligence and Multifacetted Systems Modeling Methodology [24, 29]. It aims to unify engineering design activities and develop methods for systematic simulation model construction and evaluation.

This work was funded by Siemens Corporate Research and Support Division, Princeton, New Jersey.

TRANSACTIONS of The Society for Computer Simulation
ISSN 0740-6797/90 $2.00+.10

Volume 7, Number 3, pp. 195-228

A design methodology is an important basis for supporting automation of the design process. A number of methodologies and design systems have been developed to aid the engineering design process in different domains [1, 4, 7, 9, 25]. Although different systems use different models to represent design knowledge and different systems work in different domains, there are common traits that transcend specific application domains. Knowledge-based frameworks consider design as a technological activity in which knowledge about a specific domain is used to represent design artifacts, constraints, and requirements. It is an activity that seeks all relevant knowledge and combines it to produce a design solution. Design is often considered as a search process in which a satisfactory design solution is produced from a number of alternatives [7, 19, 22, 34]. The search proceeds in a design space whose elements are design objects (components) and attributes (parameters).

The system design approach proposed by Rozenblit [19, 21], termed Knowledge-Based Simulation Design, focuses on the use of modeling and simulation techniques to build and evaluate models of the system being designed. It treats the design process as a series of activities that include: specification of design levels in a hierarchical manner (decomposition), classification of system components into different variants (specialization), selection of components from specializations and decompositions, development of design models, experimentation and evaluation via simulation, and choice of design solutions.

The design model construction process begins with developing a representation of design components and their variants. To appropriately represent the family of design configurations, we have proposed a representation scheme called the *system entity structure* (SES) [19, 29]. The scheme - which we shortly explain in more detail - captures the following three relationships: decomposition, taxonomy, and coupling. Decomposition knowledge means that the structure has schemes for representing the manner in which an object is decomposed into components. Taxonomic knowledge is a representation for the kinds of variants that are possible for an object, i.e. how it can be categorized and subclassified. The synthesis (coupling) constraints impose a manner in which components identified in decompositions can be connected together. The selection constraints limit choices of variants of objects determined by the taxonomic relations.

Beyond this, procedural knowledge is available in the form of production rules [17, 26]. They can be used to manipulate the elements in the design domain by appropriately selecting and synthesizing the domain's components. This selection and synthesis process is called *pruning* [19, 21, 22]. Pruning results in a recommendation for a *model composition tree*, i.e. the set of hierarchically arranged entities corresponding to model components. A composition tree is generated from the system entity structure by selecting a unique entity for specializations and a unique aspect for an entity with several decompositions.

The final step in the framework is the evaluation of alternative designs. This is accomplished by simulation of models derived from the composition trees. Discrete Event System Specification (DEVS) [29, 20] is a modeling formalism used

for system specification in t
of discrete event systems.
construction of hierarchical

Performance of de
the DEVS-Scheme enviro
simulation shell for modeli
models specified in the DE
with respect to experimen
Results are compared and
results in a ranking of mode
set of design objectives.

In this paper, we ill
flexible, integrated multi-uni
test and diagnosis plans, 2)
3) performance evaluation
facilities for modeling the
alternative test strategies.
material handling systems
strategies.

Our work is motiva
frameworks. Recent advanc
advances in testing [6, 8]. Ne
much faster than adequate
computer-aided circuit desi
automated test equipment s
design and control of the aut

The complexity and
the high cost of automated
frameworks. Over the last
types, component types, and
mixed-signal, Application Sp
There is no single test strate
wide spectrum of board type
must be developed which c
Test procedures and devices
We shall propose a flexi
methodology.

In what follows, we
methodology. Then, we illus
architecture.

for supporting automation of
l design systems have been
rent domains [1, 4, 7, 9, 25].
resent design knowledge and
common traits that transcend
eworks consider design as a
c domain is used to represent
activity that seeks all relevant
n. Design is often considered
n is produced from a number
design space whose elements
ters).

Rozenblit [19, 21], termed
e of modeling and simulation
being designed. It treats the
cification of design levels in a
of system components into
ents from specializations and
mentation and evaluation via

begins with developing a
. To appropriately represent
representation scheme called
- which we shortly explain in
s: decomposition, taxonomy,
the structure has schemes for
composed into components.
s of variants that are possible
subclassified. The synthesis
h components identified in
on constraints limit choices of
s.

ble in the form of production
nents in the design domain by
omponents. This selection and
g results in a recommendation
archically arranged entities
n tree is generated from the
r specializations and a unique

on of alternative designs. This
from the composition trees.
is a modeling formalism used

for system specification in the methodology. DEVS provides a formal representation of discrete event systems. It is closed under coupling. This property facilitates the construction of hierarchical DEVS network specifications.

Performance of design models is evaluated through computer simulation in the DEVS-Scheme environment [31, 32]. DEVS-Scheme is an object-oriented simulation shell for modeling and design that facilitates construction of families of models specified in the DEVS formalism. Alternative design models are evaluated with respect to experimental frames that reflect design performance questions. Results are compared and traded off in the presence of conflicting criteria. This results in a ranking of models and supports choices of alternatives best satisfying the set of design objectives.

In this paper, we illustrate the KBSD design methodology with the design of flexible, integrated multi-unit test cells. Design activities include: 1) development of test and diagnosis plans, 2) configuration and re-configuration of the test cell, and 3) performance evaluation of candidate configurations. The methodology provides facilities for modeling the test cell (hereafter called tester) and helping to explore alternative test strategies. The methodology helps to select test equipment and material handling systems. It provides methods to implement candidate test strategies.

Our work is motivated by the need to provide more flexible circuit testing frameworks. Recent advances in design of printed circuit boards have outdistanced advances in testing [6, 8]. New integrated circuits can be developed and manufactured much faster than adequate testing procedures can be devised for them. Whereas computer-aided circuit design tools are widely used in the electronic industry and automated test equipment stations are emerging, there is not sufficient support for design and control of the automated test equipment [6, 8, 12, 13].

The complexity and rapid changes in PCB manufacturing technologies and the high cost of automated test stations warrant efforts to develop flexible testing frameworks. Over the last decade, there has been a proliferation of board design types, component types, and technologies used to manufacture them. Analog, digital, mixed-signal, Application Specific Integrated Circuits (ASICs) are in widespread use. There is no single test strategy that can be the optimal approach for testing such a wide spectrum of board types and designs [12]. It is clear that new testing methods must be developed which call for fundamental changes in the approach to testing. Test procedures and devices must adapt and integrate to new board types and designs. We shall propose a flexible testing framework stemming from the KBSD methodology.

In what follows, we summarize the concepts and methods of the KBSD methodology. Then, we illustrate its application with the design of a flexible tester architecture.

2. KNOWLEDGE-BASED SIMULATION DESIGN METHODOLOGY

In this section, we provide an overview of the Knowledge-Based System Design Methodology. We discuss the system entity structure representation, the pruning procedures for generating design configurations, and the DEVS-Scheme simulation modeling layer of our framework.

2.1 Design Model Representation-System Entity Structure

As a step toward a complete knowledge representation scheme for design support we have combined the decomposition, taxonomic, and coupling relationships in a knowledge representation scheme called the *system entity structure* (SES) [29]. Knowledge representation is now generally accepted to be a key ingredient in designing artificial intelligence software. Previous work [19, 20, 21, 23] identified the need for representing the structure and behavior of systems, in a declarative scheme related to frame-theoretic and object-based formalisms [31, 32]. The elements represented are motivated, on the one hand, by systems theory [16, 27] concepts of decomposition (i.e. how a system is hierarchically broken down into components) and coupling (i.e. how these components may be interconnected to reconstitute the original system). On the other hand, systems theory has not focused on taxonomic relations, as represented for example in frame-hierarchy knowledge representation schemes. In the SES scheme, such representation concerns the admissible variants of components in decompositions and further specializations of such variants.

A system entity structure is a labeled tree. Nodes of the tree are classified as *entities, aspects, specializations,* and *multiple decompositions.* Variables can be attached to nodes. They are called *attached variables types.* An *entity* signifies a conceptual part of the system being represented by the entity structure. An *aspect* is a mode of decomposing an entity. A *specialization* is a mode of classifying an entity. An entity may have several specializations (and/or decompositions); each specialization (decomposition) may have several entities. The original entity is called a general type relative to the entities of a specialization. The entities of a specialization are called specialized types. Since each entity may have several specializations, a hierarchical structure called taxonomy results. A *multiple decomposition* is a means of representing varying number of entities. An *attached variable type* represents an attribute of an object symbolized by the entity with which the variable type is associated.

Figure 1 depicts a high level view of the entity structure representing a printed circuit board. The entity Printed Circuit Board is classified (in *functional specialization*) into Hybrid, Digital, and Analog Circuits. The *typical components decomposition* defines six major component entities: Integrated Circuits, Power Supply, Transistors, Diodes, Capacitors, and Resistors. Multiple decomposition (graphically represented in the figure by triple vertical lines) is used to represent subentities of Transistors, Diodes, Capacitors, and Resistors. Notice the presence of attached variables at various entities of the tree. They describe attributes of the objects represented by entities. For instance, Power Supply has *voltage level* and *current level.* Diode may be characterized by its *reverse* and *forward current.*

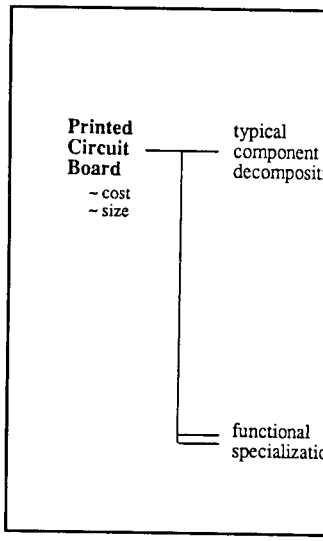


Figure 1. An Ex

Figure 1a is a refin
It illustrates several levels of
Circuit Board.

The interaction of
SES affords a compact speci
SES organizes possibilities f
a variety of model construc
definition and representatic
aggregation and abstractic
structure and its associated st

2.2 Rule-Ba

In the KBSD metho
in the model base. A synth
from the system entity stru
Pruning results in a design m
design requirements. More c
knowledge-based search thr
problem. Production rules
modeling objectives, couplir
expectations. The aim of p
optimal solution to the design
constraints).

N METHODOLOGY

e Knowledge-Based System
tructure representation, the
ons, and the DEVS-Scheme

ntity Structure

sentation scheme for design
c, and coupling relationships
n *entity structure* (SES) [29].
d to be a key ingredient in
[19, 20, 21, 23] identified the
ems, in a declarative scheme
ms [31, 32]. The elements
s theory [16, 27] concepts of
down into components) and
nnected to reconstitute the
s not focused on taxonomic
ny knowledge representation
ns the admissible variants of
s of such variants.

es of the tree are classified as
ositions. Variables can be
types. An *entity* signifies a
ntity structure. An *aspect* is a
node of classifying an entity.
/or decompositions); each
The original entity is called
ization. The entities of a
ch entity may have several
nomy results. A *multiple*
ber of entities. An *attached*
ized by the entity with which

ity structure representing a
rd is classified (in *functional*
its. The *typical components*
Integrated Circuits, Power
s. Multiple decomposition
al lines) is used to represent
tors. Notice the presence of
ey describe attributes of the
Supply has *voltage level* and
nd *forward current*.

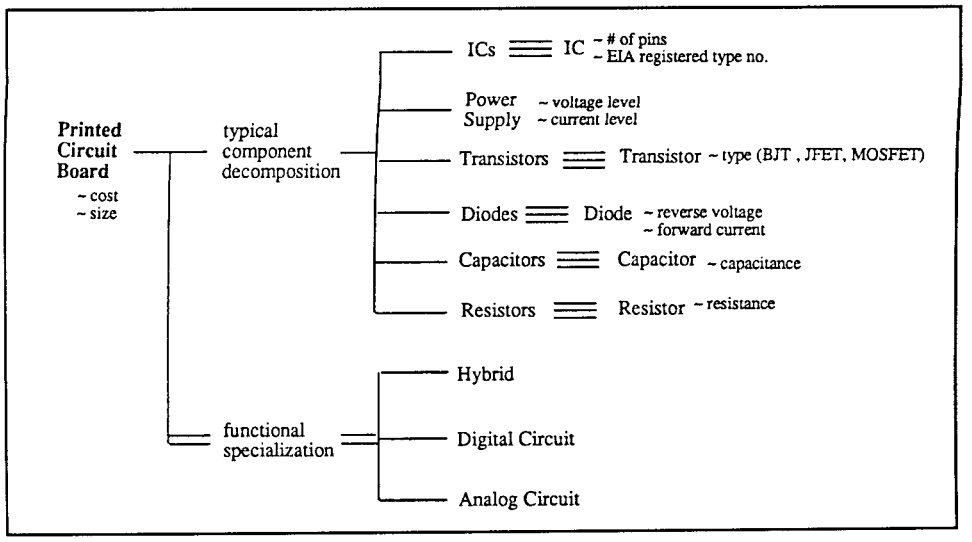


Figure 1. An Example of High Level System Entity Structure

Figure 1a is a refinement of the system entity structure depicted in Figure 1. It illustrates several levels of specializations and decompositions of a generic Printed Circuit Board.

The interaction of decomposition, coupling and taxonomic relations in an SES affords a compact specification of a family of models for a given domain. The SES organizes possibilities for a variety of system decompositions and, consequently, a variety of model constructions. Its generative capability facilitates convenient definition and representation of models and their attributes at multiple levels of aggregation and abstraction. More complete discussions of the system entity structure and its associated structure transformations are presented in [19, 21, 19, 33].

2.2 Rule-Based System Entity Structure Pruning

In the KBSD methodology, a model is synthesized from components stored in the model base. A synthesis specification is the result of *pruning* a substructure from the system entity structure. Project objectives guide the *pruning* process. Pruning results in a design model structure candidate for a best match to the set of design requirements. More detail can be found in [22]. Pruning can be viewed as a knowledge-based search through the space of candidate solutions to the design problem. Production rules are used to represent the knowledge consisting of modeling objectives, coupling constraints, user's requirements and performance expectations. The aim of pruning is to recommend plausible candidates for an optimal solution to the design problem (with respect to the design requirements and constraints).

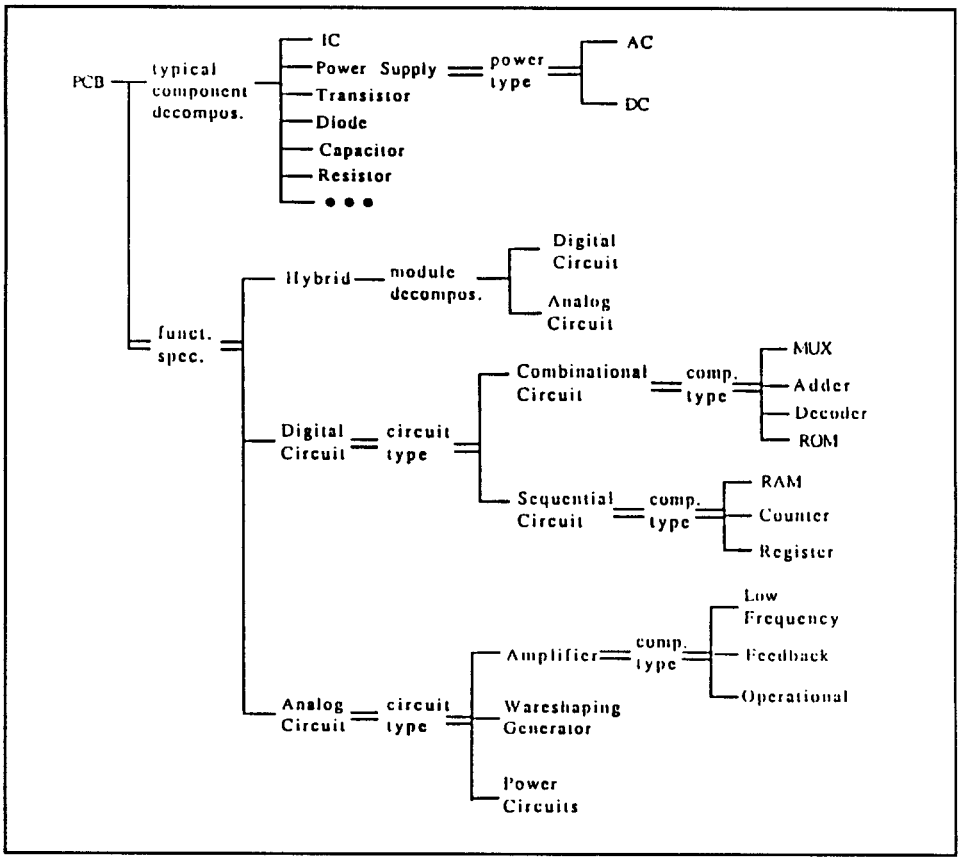


Figure 1a. System Entity Structure Representation of a Generic Printed Circuit Board

The following steps are required to provide the rules that guide pruning of the system entity structure: 1) for each specialization, specify a set of rules for selecting an entity; 2) for an entity with several aspects, specify rules for selecting a unique aspect; 3) for each aspect specify rules that ensure that the entities selected from specializations are configurable, i.e. the components they represent can be validly coupled. Thus we have two types of rules:

Selection Rule Set: selection rules are associated with entities which have children (aspects or specializations). The rules determine which specialization or aspect should be selected.

Synthesis Rule Set: synthesis rules are associated with aspects. The rules check whether the aspect's entities are configurable with respect to the coupling constraints.

Each rule can be assigned a certainty factor indicating the rule's degree of

applicability. Figure 2 illustra
and synthesis rules. We shall

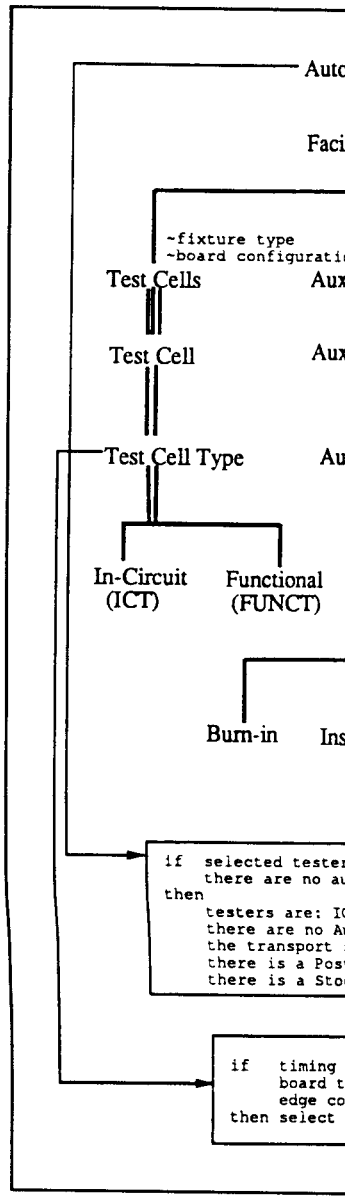
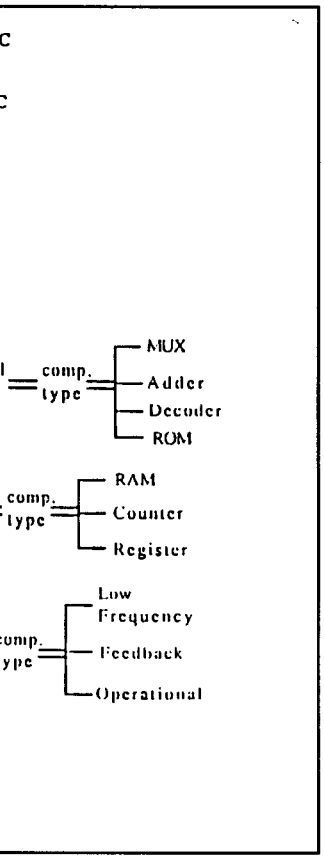


Figure 2. SES of a Test F

The above rule sets
that prunes a system entity



esentation
Board

e rules that guide pruning of
n, specify a set of rules for
specify rules for selecting a
ure that the entities selected
nents they represent can be

entities which have children
ich specialization or aspect

th aspects. The rules check
t to the coupling constraints.

dicating the rule's degree of

applicability. Figure 2 illustrates a system of a test facility, with associated selection and synthesis rules. We shall return to this example for more explanation.

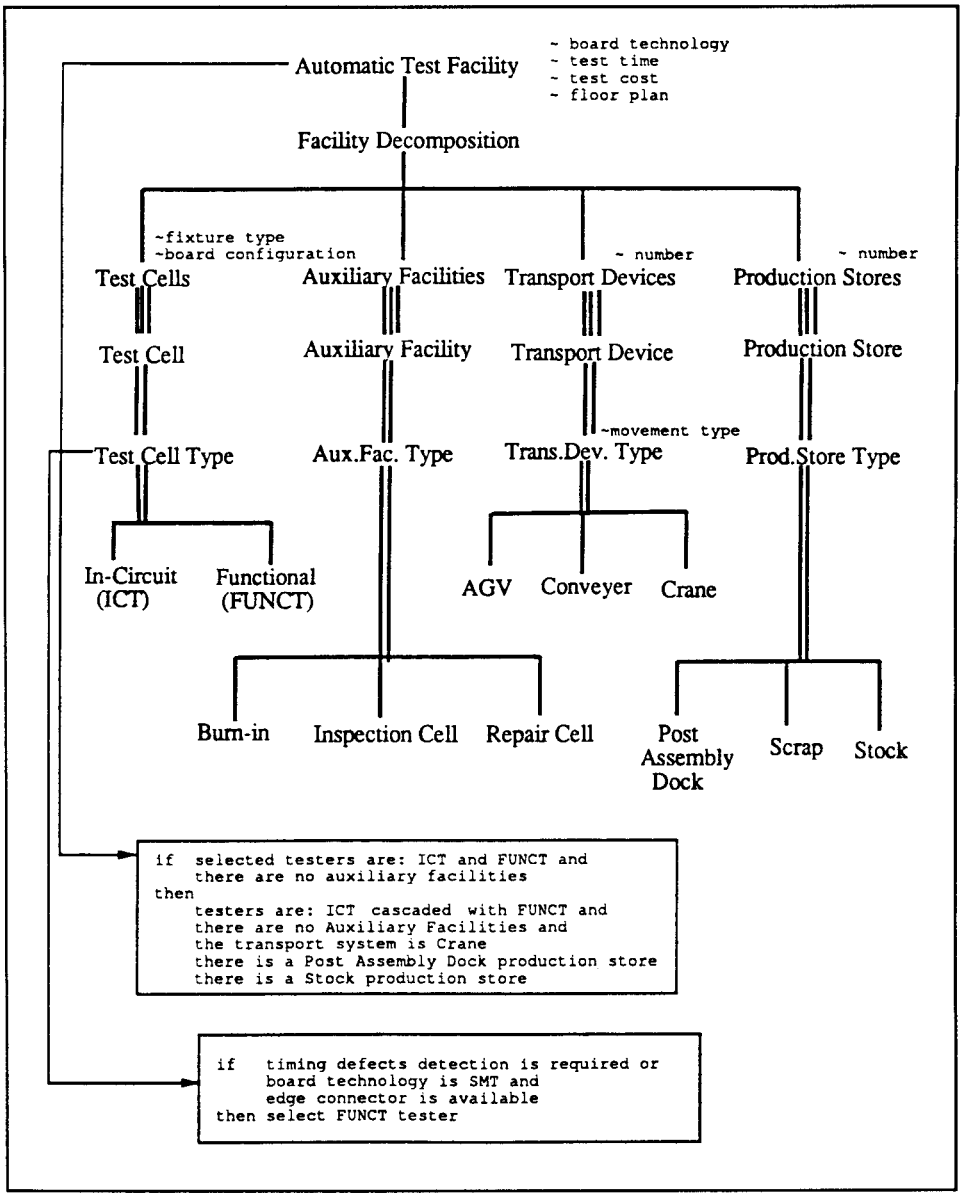


Figure 2. SES of a Test Facility with Example Selection and Synthesis Rules

The above rule sets constitute a knowledge base for the inference engine that prunes a system entity structure for a particular application domain. Pruning

results in a recommendation for a *model composition tree.* Figure 3 illustrates the composition tree concept. The model composition tree contains all the information needed to synthesize a model in a hierarchical fashion from its atomic model components.

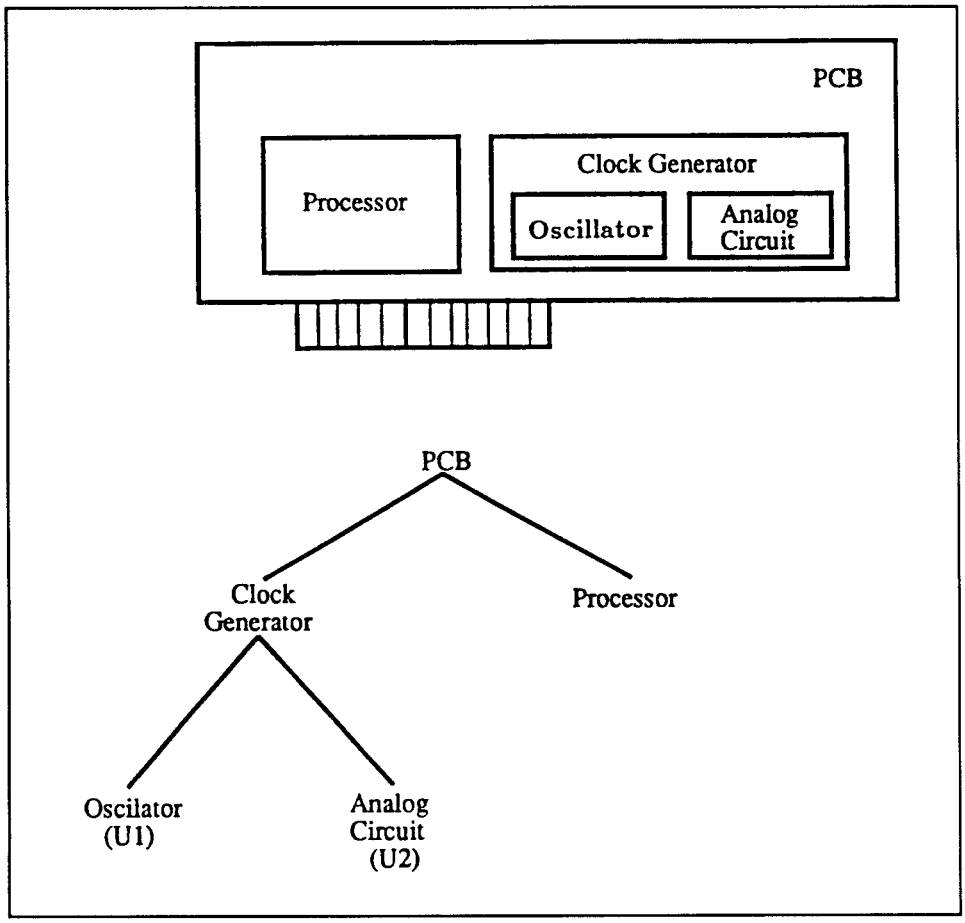


Figure 3. The Composition Tree Concept

To support the model construction process, we have available a set of software tools that are currently being integrated on AI workstations and PCs. An expert system shell MODSYN (MODel SYNthesizer) [11, 22] to generate model structures was developed and implemented. MODSYN uses selection and synthesis rules to generate a recommendation for a composition tree. A suitable simulation environment can, given a composition tree, automatically generate a model ready to simulate.

2.3 DEVS-Schem

DEVS-Scheme is
simulation models from a co
modeling and simulation laye
hierarchical, modular form
key supposition in the teste
article, we can only provide
Specification Formalism) an
available in [14, 15, 31, 32].

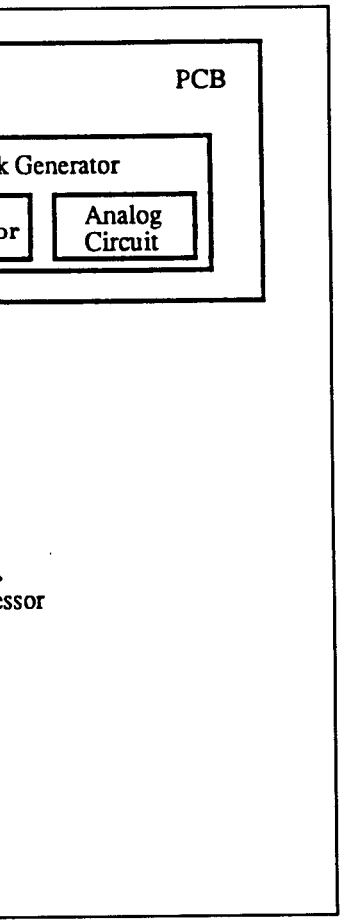
Discrete event mo
design of complex manufac
others. Long overdue recog
publication of a special issue
Yu-Chi Ho [10]. Powerful
describing such models for c
nature of discrete event
representations) is still in rela

Since the early 70's
for modeling discrete event s
concepts of Zadeh and Dos
attempted to cast both con
systems modeling framewo
publications primarily summ
concepts were an important
which simulation could be pe
for example [18]). The rec
software and hardware has
from research to practice [32].

The Discrete Event
Zeigler [29, 30] provides a me
Basically, a system has a tin
determining next states and o
provided by the DEVS form
discrete event simulation lang
this abstraction, it is possil
semantics that are easier to un

DEVS-Scheme, an
Lisp dialect), supports buildin
systems oriented approach no
such as Simscript, Simula, GA
based) or CSMP and ACSL (w

tree. Figure 3 illustrates the
contains all the information
hion from its atomic model



Concept

s, we have available a set of
AI workstations and PCs. An
r) [11, 22] to generate model
N uses selection and synthesis
n tree. A suitable simulation
ally generate a model ready to

2.3 DEVS-Scheme Modeling and Simulation Environment

DEVS-Scheme is a simulation environment that synthesizes executable simulation models from a composition tree specification [31, 32]. It thus serves as the modeling and simulation layer underpinning the KBSD methodology. Moreover, the hierarchical, modular form of model construction supported by DEVS-Scheme is a key supposition in the tester architecture application to be discussed later. In this article, we can only provide a brief review of the DEVS (Discrete Event System Specification Formalism) and its implementation in DEVS-Scheme. More detail is available in [14, 15, 31, 32].

Discrete event modeling is finding even more application to analysis and design of complex manufacturing, communication, and computer systems among others. Long overdue recognition of the importance of the field emerged with the publication of a special issue on DEDS (Discrete Event Dynamic Systems) edited by Yu-Chi Ho [10]. Powerful languages and workstations have been developed for describing such models for computer simulation. Yet, general understanding of the nature of discrete event systems *per se* (as distinct from their computer representations) is still in relative infancy compared to that of continuous systems.

Since the early 70's work has been proceeding on a mathematical formalism for modeling discrete event systems. One approach, inspired by the systems theory concepts of Zadeh and Dosoer [28], Wymore [27], Mesarovic and Takahara [16], attempted to cast both continuous and discrete event models within a common systems modeling framework. This approach was elaborated in a number of publications primarily summarized in [29] and reviewed in [30]. Systems modeling concepts were an important facet in a movement to develop a methodology under which simulation could be performed in a more principled and secure manner (see for example [18]). The recent advent of high performance artificial intelligence software and hardware has facilitated the transfer of this simulation methodology from research to practice [32].

The Discrete Event System Specification (DEVS) formalism introduced by Zeigler [29, 30] provides a means of specifying a mathematical object called a system. Basically, a system has a time base, inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs [29]. The insight provided by the DEVS formalism is in the simple way that it characterizes how discrete event simulation languages specify discrete event system parameters. Having this abstraction, it is possible to design new simulation languages with sound semantics that are easier to understand.

DEVS-Scheme, an implementation of the DEVS formalism in Scheme (a Lisp dialect), supports building models in a hierarchical, modular manner. This is a systems oriented approach not possible in popular commercial simulation languages such as Simscript, Simula, GASP, SLAM and Siman (all of which are discrete event based) or CSMP and ACSL (which are for continuous models).

Basic Models

In the DEVS formalism, one must specify 1) basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion. In this formalism basic models are defined by the structure:

$$M = < X,S,Y,\delta,\lambda,ta >$$

where X is the set of external input event types, S is the sequential state set, Y is the set of external event types generated as output, δ_{int} (δ_{ext}) is the internal (external) transition function dictating state transitions due to internal (external input) events, λ is the output function generating external events at the output, and ta is the time-advance function. Rather than reproduce the full mathematical definition here [29], we proceed to describe how it is realized in DEVS-Scheme.

To specify modular discrete event models requires that we adopt a different view than that fostered by traditional simulation languages. As with modular specification in general, we must view a model as possessing input and output ports through which all interaction with the environment is mediated. In the discrete event case, events determine values appearing on such ports. More specifically, when external events, arising outside the model, are received on its input ports, the model description must determine how it responds to them. Also, internal events arising within the model change its state, as well as manifest themselves as events on the output ports to be transmitted to other model components.

A basic model contains the following information:

- the set of input ports through which external events are received
- the set of output ports through which external events are sent
- the set of state variables and parameters: two state variables are usually present - *phase* and *sigma* (in the absence of external events the system stays in the current *phase* for the time given by *sigma*)
- the time advance function which controls the timing of internal transitions- when the *sigma* state variable is present, this function just returns the value of *sigma*.
- the internal transition function which specifies to which next state the system will transit after the time given by the time advance function has elapsed.
- the external transition function which specifies how the system changes state when an input is received - the effect is to place the system in a new *phase* and *sigma* thus scheduling it for a next internal transition; the next state is computed on the basis of the present state, the input port and value of the external event, and the time that has elapsed in the current state.
- the output function which generates an external output just before an internal transition takes place.

Coupled Models

Basic-models may
component model which is de

$$DN = < D,M_i,I_i,Z_{i,j},$$

where:

D: is a set of compo
for each i in D,

M_i: is a component

I_i: is a set, the influe
and for each j in I_i,

$Z_{i,j}$: is a function, th
and

select: is a function,

Multi-component n
models. A coupled model te
together to form a new m
component in a larger couple
coupled model contains the fo

- the set of compon
- for each compone
- the set of input po
- the set of output p

The coupling specifi

- the external input
  model to one or r
  inputs received by
- the external outpu
  to output ports of
  by a component it
  model and be tran
- the internal coupli
  ports of other com
  may be sent to th
  being sent to an ou
- the select function
  the imminent com
  is allowed to carry

A multi-component
model in the DEVS formalisn
larger multi-component mo
coupling as required for hiera

asic models from which larger
cted together in hierarchical
he structure:

e sequential state set, Y is the
δ_{ext}) is the internal (external)
rnal (external input) events, λ
he output, and ta is the time-
hematical definition here [29],
me.

uires that we adopt a different
languages. As with modular
sessing input and output ports
nediated. In the discrete event
orts. More specifically, when
d on its input ports, the model
. Also, internal events arising
st themselves as events on the
ents.

tion:

al events are received
nal events are sent
two state variables are usually
e of external events the system
n by *sigma*)
he timing of internal transitions-
t, this function just returns the

pecifies to which next state the
y the time advance function has

pecifies how the system changes
t is to place the system in a new
next internal transition; the next
present state, the input port and
that has elapsed in the current

n external output just before an

Coupled Models

Basic-models may be coupled in the DEVS formalism to form a multi-component model which is defined by the structure:

$$DN = < D, M_i, I_i, Z_{i,j}, SELECT >$$

where:

D: is a set of component names;

for each i in D,

M_i: is a component basic model

I_i: is a set, the influencees of i

and for each j in I_i,

$Z_{i,j}$: is a function, the i-to-j output translation

and

select: is a function, the tie-breaking selector.

Multi-component models are implemented in DEVS-Scheme as coupled models. A coupled model tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction. A coupled model contains the following information:

- the set of components
- for each component, its influencees
- the set of input ports through which external events are received
- the set of output ports through which external events are sent.

The coupling specification consists of:

- the external input coupling which connects the input ports of the coupled model to one or more of the input ports of the components - this directs inputs received by the coupled model to designated component models
- the external output coupling which connects output ports of components to output ports of the coupled model - thus when an output is generated by a component it may be sent to a designated output port of the coupled model and be transmitted externally
- the internal coupling which connects output ports of components to input ports of other components - when an input is generated by a component it may be sent to the input ports of designated components (in addition to being sent to an output port of the coupled model)
- the select function which embodies the rules employed to choose which of the imminent components (those having the minimum time of next event) is allowed to carry out its next event.

A multi-component model DN can be expressed as an equivalent basic model in the DEVS formalism [29]. Such a basic model can itself be employed in a larger multi-component model. This shows that the formalism is closed under coupling as required for hierarchical model construction.

Testability of Hierarchical, Modular Models

An important benefit of the object-oriented, hierarchical, and modular nature of DEVS-Scheme is its support for model verification. We briefly review the testing process it facilitates as a basis for our later formulation of testing architecture concepts.

Testability: Models, as objects, can be tested against their behavioral requirements specification (e.g. given in axiomatic form) by injecting sequences of messages and comparing their response with that expected.

Modularity: Model specifications are self-contained and have input and output ports through which all interaction with the external world must take place. In addition, ports provide a level of delayed binding which needs to be resolved only when models are coupled together.

Stand-alone and Bottom-up Testability: Due to object encapsulation and input/output modularity, models are independently verifiable at every stage of hierarchical construction. This fosters secure and incremental bottom-up synthesis of complex models.

To summarize, the DEVS formalism underlies DEVS-Scheme, a general purpose environment for constructing hierarchical, modular discrete event models. DEVS-Scheme is written in the PC-Scheme language which runs on DOS compatible microcomputers and under a Scheme interpreter for the Texas Instruments Explorer. DEVS-Scheme is implemented as a shell that sits upon PC-Scheme in such a way that all of the underlying Lisp-based and objected oriented programming language features are available to the user. The result is a powerful basis for combining AI and simulation techniques.

2.4 Phases in the KBSD Methodology

We now outline the phases required to execute the KBSD methodology.

1. We begin model construction process by conceptualizing decompositions and specializations of components of the system being modeled. To do this, we use system entity structuring tools. We utilize the system entity structure base as a repository of previous modeling experience. Thus, we may retrieve an entity structure from this base which is applicable to the modeling domain at hand. Such an entity structure is modified and enhanced with entities required in the new project. Models associated with new atomic entities must be developed and placed in the model base.

2. We develop a rule base to be used in the pruning process. We construct the rule base from the rules already associated with the system entity

structure at ha
constraints and

3. We invoke the
candidate solu
composition tre

4. We invoke the
the composition

5. To carry out
experimental f
observe the be
DEVS-Scheme
model, i.e. discr
c) control the s
detailed present
20, 29].

6. We evaluate si
performance m

The above phases m

The system entity st
be illustrated in the following
rapidly develop and evaluate
Such evaluation culminates in
such performance criteria a
inventory, etc. [2, 3]. We no
detail.

3. T

As illustrated in Fig
following phases and p
strategy/architecture combin
The system under test is r
representation is used in conj
level. The fault modeling
developed for the purpose o
Printed Circuit Boards (PCBs
DEVS models with fault lo
architecture and testing strat
the configuration of the test
representation of a tester. T
DEVS-Scheme simulation
strategies can be compared t
construction of a test cell. We

structure at hand. We also create new rules based on the coupling constraints and design requirements.

3. We invoke the pruning engine to generate recommendations for candidate solutions to the design problem in the form of model composition trees.

4. We invoke the transformation procedure that synthesizes models from the composition trees obtained in phase 3.

5. To carry out a simulation experiment, we need to specify an experimental frame, i.e. the set of circumstances under which we observe the behavior of the model. We accomplish this by defining DEVS-Scheme components that: a) generate input stimuli to the model, i.e. discrete event input segments; b) observe model output; and c) control the simulation experiment by observing model variables. A detailed presentation of the experimental frame concepts is given in [19, 20, 29].

6. We evaluate simulation results and rank models with respect to the performance measures that express design objectives and requirements.

The above phases may be iterated in a feedback process.

The system entity structure, DEVS formalism, and experimental frames will be illustrated in the following design methodology. We will show how they help to rapidly develop and evaluate alternative design models of test systems and strategies. Such evaluation culminates in the selection of a design that is optimal with respect to such performance criteria as test/rework time, first pass yield, work in process inventory, etc. [2, 3]. We now proceed to describe the testing methodology in more detail.

3. TESTING METHODOLOGY

As illustrated in Figure 4, the postulated testing framework consists of the following phases and processes. First, performance criteria for test strategy/architecture combinations and associated experimental frames are defined. The system under test is represented using the system entity structure. This representation is used in conjunction with test criteria for selection of a test resolution level. The fault modeling process is supported by DEVS models specifically developed for the purpose of modeling fault locations in the system under test. Printed Circuit Boards (PCBs) are discussed to illustrate the applicability of concepts. DEVS models with fault locations represent faulty PCBs for which a test station architecture and testing strategy are being designed. The test strategy selection and the configuration of the test station are generated from the generic entity structure representation of a tester. The tester architecture is modeled and simulated using DEVS-Scheme simulation language. Alternative tester architectures and test strategies can be compared through a simulation modeling study prior to an actual construction of a test cell. We now characterize each phase of the methodology.

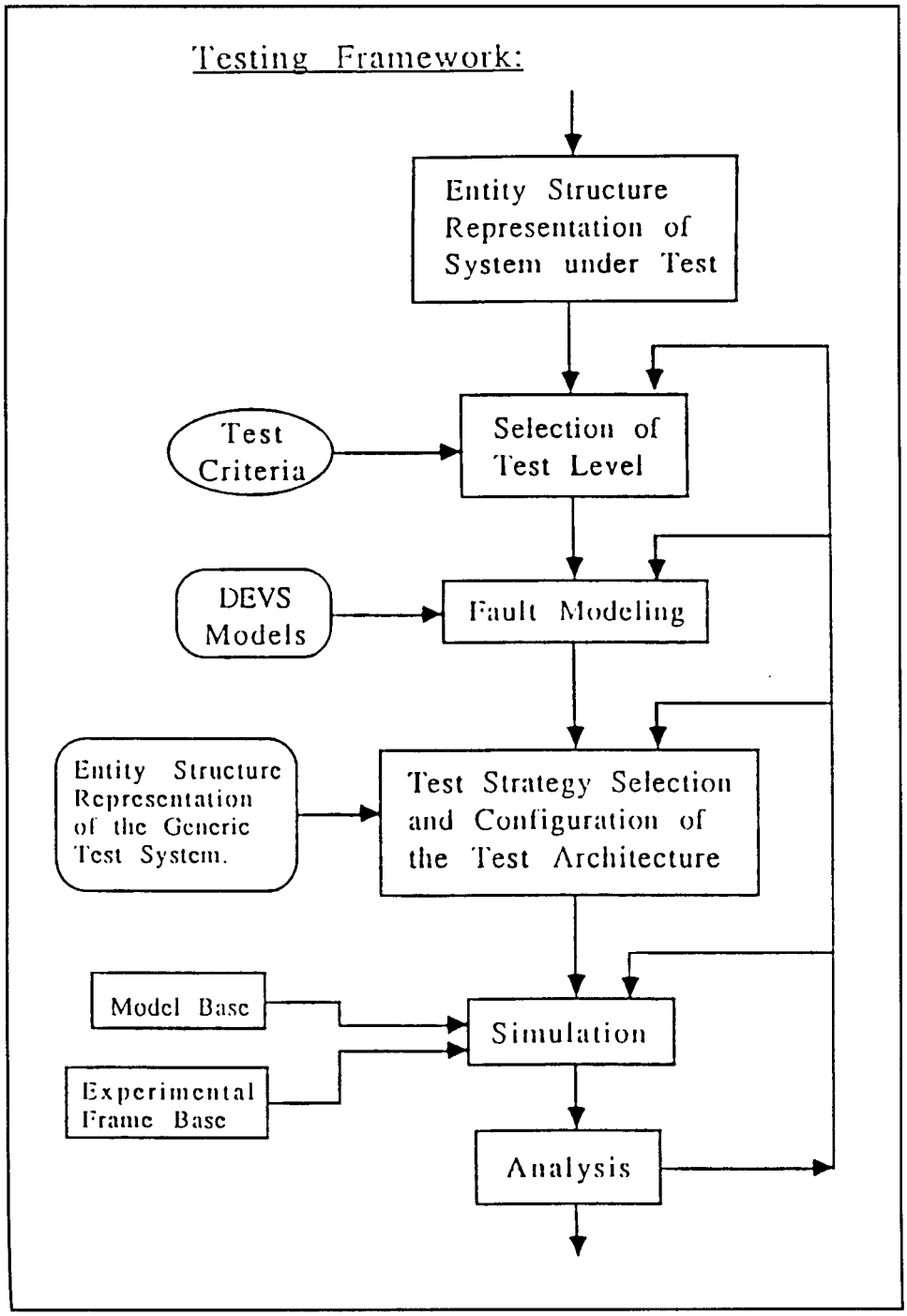


Figure 4. Postulated Testing Framework

3.1 Experime

We separate the m
which the model is observed.
ranking and relieves the mo
themselves.

A set of circumstan
experimented with is called
experimental frame can be re
(supplying a model with an
environment upon a model),
a transducer (collecting and
experimental frames in the
specifying basic models and th

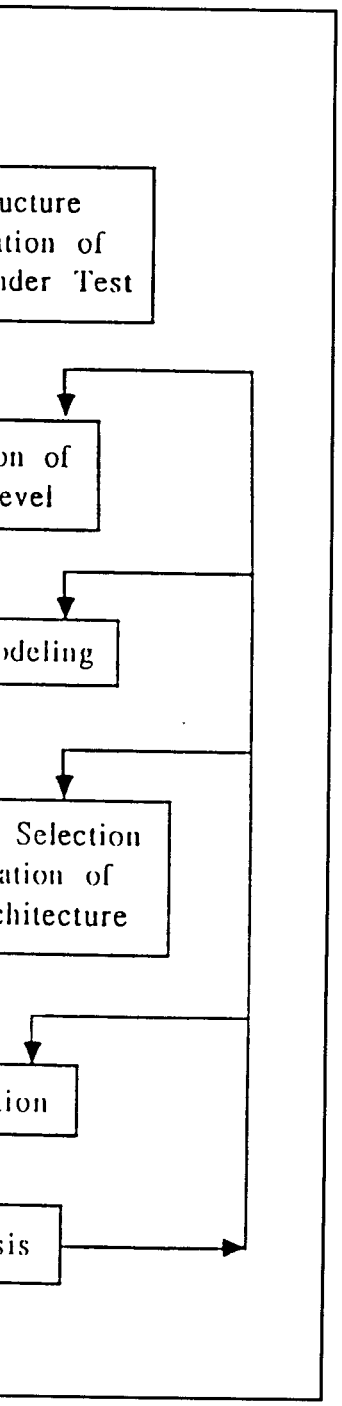
Experimental frame
tester performance criteria
detected per lot, average test
test cells' utilization, time to r

Inputs to a model
modeled at a specific locat
distribution. Outputs will be
above. The experiment will
percent of faults detected, wo
work-in-process exceeds a cert

3.2 Entity Struct

We employ the syste
under test, its decompositions
representation may appear
designers tend to view a board
components as being arranged
a functional specification per
contain sub-blocks which in
functional specifications arises
system entity structure repres
model construction is the basi
tester architecture.

Figure 1, to which w
entity structure representatio
would consist of a subset of c
structure. For example, the to
a PC system board for IBM XT



ework

3.1 Experimental Frames and Performance Criteria

We separate the model description from a simulation experiment under which the model is observed. This facilitates much greater flexibility in design model ranking and relieves the models from the responsibility of collecting data about themselves.

A set of circumstances under which a model system is to be observed and experimented with is called an *experimental frame*. Zeigler [29] has shown that an experimental frame can be realized as a coupling of three components: a generator (supplying a model with an input segment reflecting the effects of the external environment upon a model), an acceptor (a device monitoring a simulation run), and a transducer (collecting and processing model output data). The specification of experimental frames in the DEVS-Scheme environment is equivalent to that of specifying basic models and their corresponding couplings.

Experimental frames reflect I/O performance design requirements. The tester performance criteria will include the following measures: percent fault detected per lot, average test time per board/lot, average work in process, individual test cells' utilization, time to repair, first pass yield.

Inputs to a model of tester configurations will be boards with a fault modeled at a specific location arriving at the tester with a given inter-arrival distribution. Outputs will be defined by the types of performance indices specified above. The experiment will be controlled by a set of run control variables such as percent of faults detected, work-in-process. For example, we may terminate a run if work-in-process exceeds a certain limit value.

3.2 Entity Structure Representation of System under Test

We employ the system entity structure to represent components of a system under test, its decompositions and taxonomic relations (specializations). This type of representation may appear controversial for printed circuit boards since most designers tend to view a board as a structure with no hierarchy, i.e. they perceive the components as being arranged on the board at the same design level. However, from a functional specification perspective, a board can be decomposed into blocks that contain sub-blocks which in turn include other blocks, etc. Thus, a hierarchy of functional specifications arises whose characteristic can be easily reflected through the system entity structure representation. As we shall show, hierarchical and modular model construction is the basis for devising flexible test strategies and designing the tester architecture.

Figure 1, to which we referred in Section 2.1, depicts a high level system entity structure representation of a printed circuit board. A specific board design would consist of a subset of components laid out by the generic board system entity structure. For example, the top level segment of the system entity structure depicting a PC system board for IBM XT Personal Computer is shown in Figure 5.

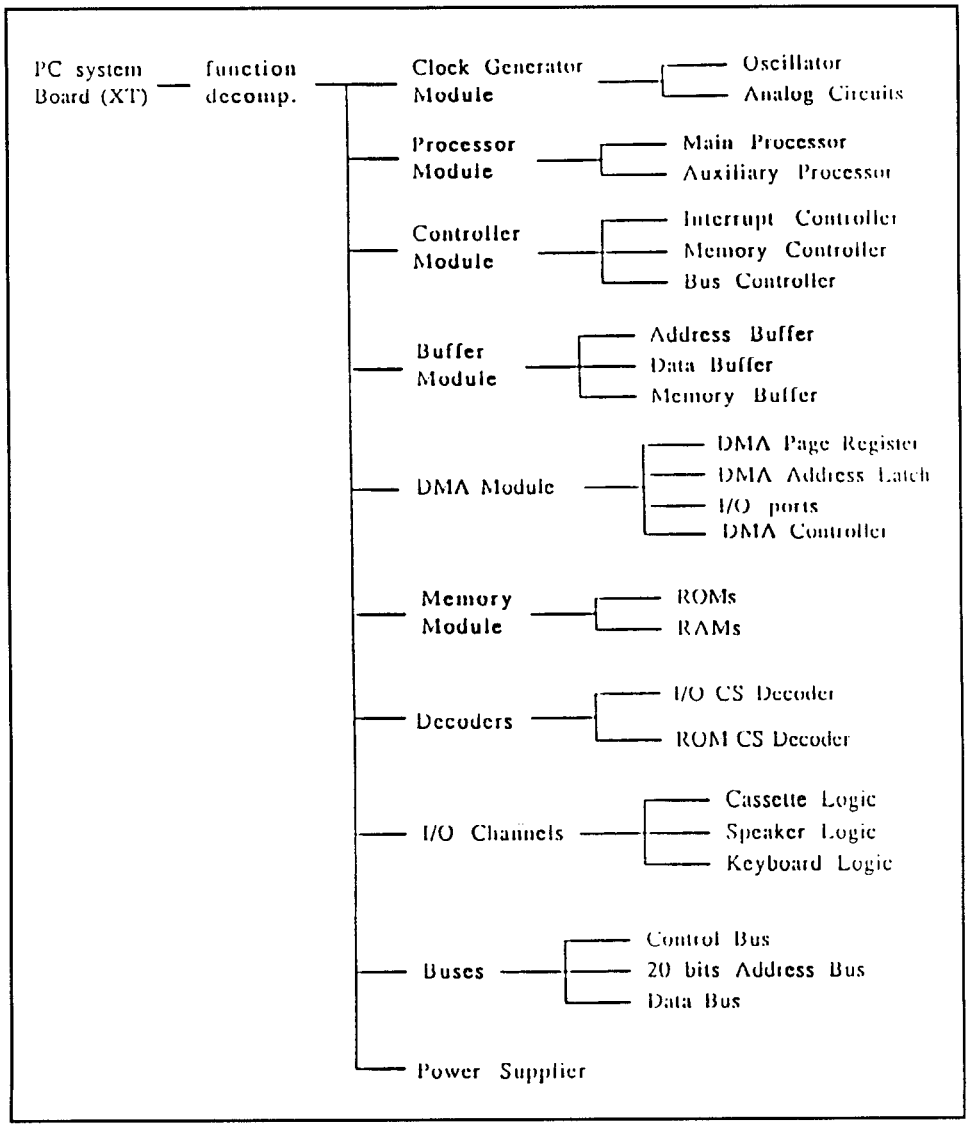


Figure 5. Top Level Segment of the System Entity Structure Structure Representation of the IBM XT System Board.

Based on the system entity structure representation, we determine the test strategy for a given board. Determining the test strategy will involve the selection of the test level and configuration of the tester architecture.

3.3 Selection of Test Level

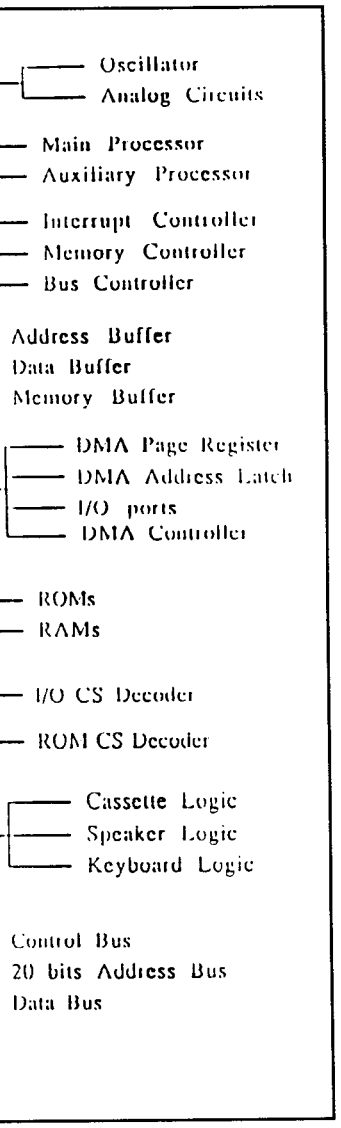
The level of testing detail is a design parameter that may critically impact

fault detectability, tester arc
level can be represented as th
is performed. One factor
Repairable Units (LRU) [32
Clock, Generator and Process
as follows:

If CLOCK module is
and (other conditions
then test level = dep
else test level = dept

Thus if the clock su
replaced or individually rep
subcomponents. Otherwise,
spend time testing the submo
location, necessitates the repl
actually proceeds to the level
of additional considerations (s
depend on test criteria elicite
worth the additional testing r
replaced with little cost. I
cost/benefit ratios. Such
component (both as a type
following question: "After
continue testing and repair
projected time-to-failure of
designed to support the dat
required in establishing such
make such levels accessible to

The next phase in ou
structures representing board
objective here is to represent,
procedures typically employ
impact on tester performa
recognizes that reproducing s
in terms of model developme
viewed as if it were lying on
path distance from the root,
level. The leftmost node at
column value n - 1, where n
tree has the same number of
num-branches. To simplify o
structures of boards under tes



Entity Structure
3M XT System Board.

ntation, we determine the test
gy will involve the selection of
e.

eter that may critically impact

fault detectability, tester architecture complexity and tester performance. The test level can be represented as the depth in the composition tree below which no testing is performed. One factor in determining test level is the existence of Least Repairable Units (LRU) [32]. For example, for a PCB module which consists of Clock, Generator and Processor as shown in Figure 3, the test level can be established as follows:

If CLOCK module is an LRU
and (other conditions hold)
then test level = depth(CLOCK) + 1
else test level = depth(CLOCK)

Thus if the clock subcomponents are repairable (meaning that they can be replaced or individually repaired), testing may proceed down to the level of those subcomponents. Otherwise, i.e. if the clock submodules are LRUs, we need not spend time testing the submodules since any fault in the clock module regardless of its location, necessitates the replacement of the whole clock unit. Whether or not testing actually proceeds to the level of a repairable component may be determined by a host of additional considerations (shown in the rule for CLOCK as (other conditions)) that depend on test criteria elicited from the test engineers. For example, it may not be worth the additional testing required to isolate the fault in a component if it can be replaced with little cost. In general, we must provide criteria in the form of cost/benefit ratios. Such ratios may also take into account the *history* of the component (both as a type and an instance) involved. For example, consider the following question: "After how many repairs does it become too expensive to continue testing and repairing this CLOCK given its history of repairs and the projected time-to-failure of CLOCK modules?" Of course the test cell must be designed to support the data collection necessary to compile histories of the sort required in establishing such break-even levels and the communication necessary to make such levels accessible to test sites.

3.4 Fault Modeling

The next phase in our testing framework is modeling of faults using the tree structures representing board components and their decompositions. Note that the objective here is to represent, at a high level of abstraction, the characteristics of test procedures typically employed in actual testing for the purpose of studying their impact on tester performance. Multifacetted Modeling Methodology explicitly recognizes that reproducing such test procedures in great detail would be prohibitive in terms of model development and simulation complexity [32]. A composition tree is viewed as if it were lying on a plane (Figure 6). Each node has a *level*, which is its path distance from the root, and a *column*, which is its order within all nodes at its level. The leftmost node at any level has column value 0; the rightmost node has column value n - 1, where n is the number of nodes at the node's level. A uniform tree has the same number of son nodes at each non-leaf node, given by parameter, *num-branches*. To simplify our initial explorations, we assume that the composition structures of boards under test are uniform trees.

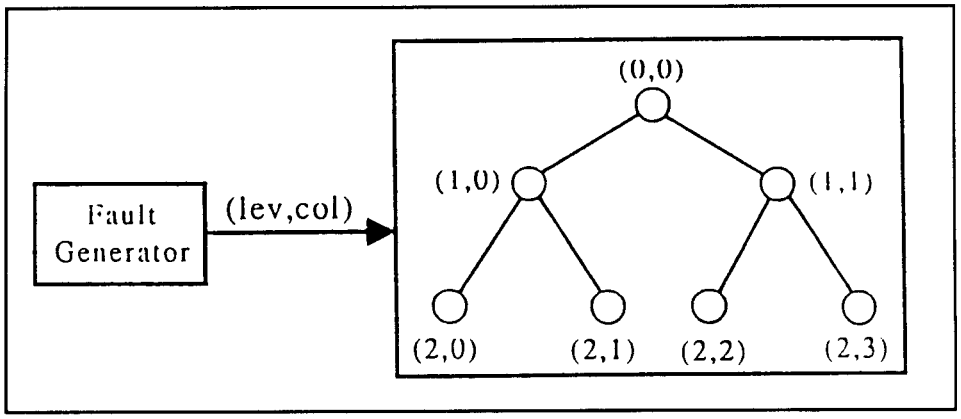


Figure 6. Tree Representation for Fault Modeling

Generating Faults

As part of the experimental frame mentioned above we use a generator to output a stream of faulty and fault-free board models. Such board models may contain as much information about boards under test as necessary to achieve the objectives of the simulation study. In the simplest case, where all boards have the same composition tree, a fault is specified merely by indicating which sub-module is faulty. In this case, the fault location is specified by a level and column of the composition tree of the board under test. For example, a fault location (2, 3) indicates a fault occurring at the component level 2 and at column 3 in the planar picture of the tree.

It is important to state the interpretation of fault location assignments. The occurrence of a fault at location (i, j) has the following consequences:

- the fault is detectable by testing at any location which is an ancestor of (i, j) in the tree. This means that the effect of fault in a component will manifest itself in all sub-systems in which it is contained. However, the probability of detecting a fault buried deep within a system may be much less than that of detecting it by testing the faulty component itself. We return to modeling this situation soon.

- the fault is detectable at location (i, j) itself.

- the fault is not detectable at any location in the subtree under (i, j). This means that all the components which are contained within the module at level (i, j) are fault free and therefore that the fault lies in the coupling of components at the next lower level, i + 1.

To generate faults we use the following algorithm with parameters *pfault,* the probability of having a fault, and *pnext-lev*, the probability that the fault is at the next

lower level from the current nod

Fault Generation Algorithm

Sample a random number and c
the current board.

If there is a fault, recursively per
At location (i, j): *Sample* a rand
If the integer, b, is less than nu
b'th oldest child of (i, j);
repeat this algorithm at this loc
whether the fault is there or at th
otherwise the fault is at (i, j).

(Termination condition
for simplicity.)

Note that *pfault*, whic
generator output stream, can
been established, the actual d
parameter *pnext-lev.* A high valu
depth of the composition tree.

Modeling Test Time

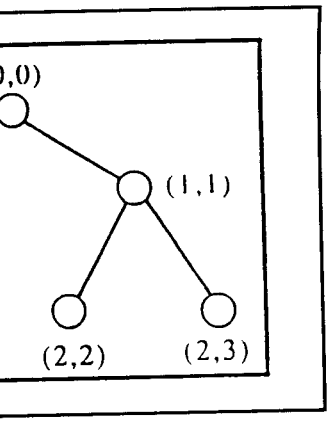
We assume that test ti
must be considered to cover a
covering them. To represent the
reliability. This is the probabil
same level at which it is located.
distribution whose mean is inver

mean test-time-r
where T is scale factor depend
used.

This represents the ass
test cases must be performed as

Modeling Fault Detection

The following algorith
probability of detecting a fault wh



lt Modeling

above we use a generator to
ls. Such board models may
st as necessary to achieve the
se, where all boards have the
ndicating which sub-module is
y a level and column of the
mple, a fault location (2, 3)
nd at column 3 in the planar

ult location assignments. The
consequences:

tion which is an ancestor of (i,
t of fault in a component will
it is contained. However, the
within a system may be much
faulty component itself. We

n the subtree under (i, j). This
contained within the module at
the fault lies in the coupling of

thm with parameters *pfault,* the
ility that the fault is at the next

lower level from the current node.

Fault Generation Algorithm

Sample a random number and compare it to *pfault* to determine if there is a fault in the current board.

If there is a fault, recursively perform the following starting with location (0, 0):
At location (i, j): *Sample* a random integer between 0 and num-branches/*pnext-lev*
If the integer, b, is less than num-branches, then it indicates a potential fault at the b'th oldest child of (i, j);
repeat this algorithm at this location which is (i+1, b+j * num-branches) to find out whether the fault is there or at the next level.
otherwise the fault is at (i, j).

(Termination conditions which test for various boundary cases are omitted for simplicity.)

Note that *pfault*, which determines the frequency of faulty boards in the generator output stream, can be selected independently. Once this frequency has been established, the actual distribution of fault locations is determined by the parameter *pnext-lev.* A high value of the latter will tend to skew faults towards greater depth of the composition tree.

Modeling Test Time

We assume that test time varies directly with the number of test cases that must be considered to cover all faults and thoroughness of the test procedure in covering them. To represent the latter coverage, we specify a parameter, desired *test-reliability*. This is the probability of detecting an existing fault while testing at the same level at which it is located. The time required for testing is then sampled from a distribution whose mean is inversely related to the *test-reliability:*

$$\text{mean test-time-required} = -T \log(1 - \text{test-reliability})$$

where T is scale factor depending on the particular test equipment and procedure used.

This represents the assumption that a combinatorially explosive number of test cases must be performed as the *test-reliability* increases toward unity.

Modeling Fault Detection

The following algorithm is used to determine prob-of-detecting-fault, the probability of detecting a fault when a particular board is tested at a test-station.

if the board under test is fault free, prob-of-detecting-fault = 0
otherwise: let the location of the fault be (f_i,f_j) and the test be carried out on a component at location (c_i,c_j);
if (f_i,f_j) is not in the subtree of (c_i,c_j)
then prob-of-detecting-fault = 0
otherwise
if $(c_i,c_j) = (f_i,f_j)$
then prob-of-detecting-fault = *test-reliability*
otherwise prob-of-detecting-fault = *test-reliability*$/(e^{(c_i - f_i)} *$ num-branches)

These rules reflect the interpretation given above concerning the assignment of locations to faults: when testing a component, a fault can be detected only if it is located within the component, either at the same level or within some sub-module. In the former case, the detection probability is by definition the *test-reliability*. In the latter case, the fault is at some non-zero depth $(c_i - f_i)$, within the component and we assume that its probability of detection decreases exponentially with this depth. This reflects the chance that tests will be sensitive to the particular faulty sub-module in distinction to all other sub-modules at the same depth.

3.5 Tester Architecture Configuration

To configure the tester architecture, we employ a second system entity structure and rule-based pruning (recall Figure 2). The system entity structure represents a flexible testing system that includes generic test stations. Given a specific mix of systems to test (PCBs), we need to select a subset of components and test procedures that are sufficient and necessary to realize the testing strategies under consideration. To support this selection process we employ rule-based pruning methodology and the pruning expert system shell MODSYN (Section 2.2).

In our testing methodology, the rules must reflect the kind of testing that is required and the test cells which must be selected in order to realize the test strategy. To illustrate this concept, let us consider an example of generating the test system proposed by Anderson [2, 3].

3.5.1 Example - Configuration of a Test Facility

Flexible testing of printed circuit boards requires that an Automatic Test Facility (ATF) be designed. The facility should have devices that are configured for testing a specific type of board. Different configurations of the ATF may be generated, depending on the type of boards being tested.

As illustrated in Figure 2, the major subcomponents (workstations) of the ATF include: test cells, transport devices, production stores, and auxiliary facilities. A test cell can be in in-circuit tester or a functional tester. A transport device can be: a conveyer, a crane, or an automatic guided vehicle (agv). A production store can be: a post assembly dock, a scrap store, or a stock store. A burn-in, an inspection cell, and a repair workstation are auxiliary facilities.

Flexible testing would co
(from the set of the ATF's comp
example, in order to test a bare-bo
without any attached devices), on
opens may be sufficient. On the
using all of the ATF's components

An explanation of the c
An in-circuit tester tests each sep
device and sensing the results fro
complete, functional entity by app
edge connector.

Transport devices are u
A production store is where the
repaired, then it is sent to a scra
dynamical operation at an elev
examine a board after it has b
rebuild the board on the site of t

Here are some fund
components. In-circuit testers
isolation of devices under test.
in order to perform an in-circu
whereas the functional tests do.
in auxiliary facility is used to ope
Such a facility is usually followed

Given the above descri
different configurations of the A
test design attributes, prune a s
Different attributes will gener
provide two different examples
pruned.

Pruning Rule Base

We provide an exam
alternative configurations of th
Appendix.

Example of Selection Rules

/*Selection of Tester Cells*/

if access to all circuit nodes on

cting-fault = 0
and the test be carried out on

$y/(e^{(c_i - f_i)} *$ num-branches)

ve concerning the assignment
t can be detected only if it is
within some sub-module. In
on the *test-reliability*. In the
within the component and we
entially with this depth. This
rticular faulty sub-module in

ation

ploy a second system entity
The system entity structure
test stations. Given a specific
bset of components and test
the testing strategies under
employ rule-based pruning
YN (Section 2.2).

ect the kind of testing that is
er to realize the test strategy.
of generating the test system

st Facility

uires that an Automatic Test
evices that are configured for
rations of the ATF may be

onents (workstations) of the
stores, and auxiliary facilities.
r. A transport device can be:
). A production store can be:
A burn-in, an inspection cell,

Flexible testing would consist in generating a configuration of components (from the set of the ATF's components) for testing a particular type of PCB. For example, in order to test a bare-board (board with circuit connections etched on it but without any attached devices), only an inspection facility that checks for shorts and opens may be sufficient. On the other hand, another type of board may be tested using all of the ATF's components.

An explanation of the components of the ATF and their functions follows: An in-circuit tester tests each separate device on a board by applying test signals to a device and sensing the results from its output. A functional tester tests a board as a complete, functional entity by applying inputs and sensing outputs through the board's edge connector.

Transport devices are used to move boards from one workstation to another. A production store is where the boards are held. For example, if a board cannot be repaired, then it is sent to a scrap store. A burn-in facility is used to test the board's dynamical operation at an elevated temperature. An inspection facility is used to examine a board after it has been tested or burned-in. A repair facility is used to rebuild the board on the site of the ATF.

Here are some fundamental criteria for selecting various kinds of components. In-circuit testers require access to all circuit nodes on the board and isolation of devices under test. A special fixture called bed-of-nails is also mandatory in order to perform an in-circuit test. In-circuit tests do not detect timing defects whereas the functional tests do. The latter tests require an edge connector. A burn-in auxiliary facility is used to operate a board dynamically at an elevated temperature. Such a facility is usually followed by inspection of the board.

Given the above description, we define a rule base that can be used to prune different configurations of the ATF. The problem here is: given a set of parameters, test design attributes, prune a set of workstations from which ATF will be composed. Different attributes will generate different arrangements of the ATF. We then provide two different examples of ATF configurations and show how they have been pruned.

Pruning Rule Base

We provide an example of selection and synthesis rules used to generate alternative configurations of the ATF. The complete knowledge base is given in the Appendix.

Example of Selection Rules

/*Selection of Tester Cells*/

if access to all circuit nodes on the UUT is available and

devices under test can be isolated and
bed-of-nails fixture is available for the PCB
timing defects detection is not required
then select ICT tester

if access to all circuit nodes on the UUT is available and
devices under test can be isolated and
bed-of-nails fixture available for the PCB and
timing of detects detection is required and
edge connector is available
then
selected testers are: ICT and FUNCT

Example of Synthesis Rules

if selected testers are: ICT and FUNCT and
All Auxiliary facilities are required
then
testers are ICT cascaded with FUNCT and
Burn-in facility is cascaded with FUNCT and
there is Inspection facility
there is Repair facility
the transport system is one Crane, one AGV and
one Conveyer
there is Post Assembly Dock production store
there is Stock production store
there is Scrap production store

if selected testers are: ICT and FUNCT and
there are no auxiliary facilities
then
testers are: ICT cascaded with FUNCT and
there are no Auxiliary Facilities and
the transport system is Crane
there is Post Assembly Dock production store
there is Stock production store

Notice that in specifying the rules, we have focused mainly on the test cell selection. The above example may be further refined by adding blocks of rules explicitly governing the selection of a transport system or repair facility type.

To illustrate the pruning process, consider a MODSYN consultation session. The user answers system queries and instantiates the objects' attributes specified in the production rules. An example of a consultation session is shown below:

Is the board a bare-board? *no*
What is the board's assembly technology? *conventional*

Can devices under test be isolat
Is the bed-of-nails fixture availa
Is timing defects detection requi
Is an edge connector available f
Is dynamical operation at elevat
Is on-site repair desired? *yes*

The resulting *Configuration Rec*

Testers are: ICT cascaded with
Burn-in facility is cascaded with
There is Inspection facility
There is Repair facility
The transport system is: one C
There is Post Assembly Dock
There is Stock production store
There is Scrap production store

The result of the prun
of cells that perform In-cir
overhead carrier, a conveyer,
used as Production Stores.
workstation. Figure 7 represe
illustrated in Figures 8 and 9
model depicted in Figure 8 w
testing at elevated temperature
test procedure. Figure 9 illust
tests are performed.

3.

The above model can
of the tester it represents.
frames in assessing tester pe
strategies that route boards th
on the basis of the test criteri
For example, the coupling con
in the following traversal rule:

if A is a "front end" to B
then test A first.

This sends the PCB
test for module B. Note, ho
the workload at the test site
sites in the routing.

Can devices under test be isolated? *yes*
Is the bed-of-nails fixture available? *yes*
Is timing defects detection required? *yes*
Is an edge connector available for this board? *yes*
Is dynamical operation at elevated temperature desired? *yes*
Is on-site repair desired? *yes*

The resulting *Configuration Recommendation* is

Testers are: ICT cascaded with FUNCT
Burn-in facility is cascaded with FUNCT
There is Inspection facility
There is Repair facility
The transport system is: one Crane, one AGV, and one Conveyer
There is Post Assembly Dock
There is Stock production store
There is Scrap production store

The result of the pruning process is depicted in Figure 7. The tester consists of cells that perform In-circuit, Functional, and Burn-in tests. There are: an overhead carrier, a conveyer, and an AGV transporter. Dock, Stock, and Scrap are used as Production Stores. There are also an Inspection facility and a Repair workstation. Figure 7 represents a candidate model for the ATF design. Others - illustrated in Figures 8 and 9 - differ in test cell types selected. For example, the model depicted in Figure 8 was generated in a consultation in which no dynamical testing at elevated temperature was necessary nor was the on-site repair critical to the test procedure. Figure 9 illustrates a model of a configuration in which no functional tests are performed.

3.6 Test Strategy: Routing

The above model can be simulated in order to obtain performance measures of the tester it represents. As discussed above, we employ suitable experimental frames in assessing tester performance. First, however, we need to consider test strategies that route boards through test stations. The traversal algorithm is defined on the basis of the test criteria, selection of test level, and SES coupling constraints. For example, the coupling constraint "module A is a front end to module B" will result in the following traversal rule:

if A is a "front end" to B
then test A first.

This sends the PCB under test to be tested for module A prior to sending to test for module B. Note, however, that other tests may intervene as determined by the workload at the test site for module B relative to the workload at other possible sites in the routing.

To study alternative routing strategies in a systematic manner, we extended the underlying simulation environment. In DEVS-Scheme, kernel-models and its subclasses provide convenient facilities for constructing models having arbitrary numbers of components, generated from a prototype, the kernel, and coupled in a uniform manner. The coupling specification is determined by a parameterized formula characteristic of each sub-class of kernel-models. Thus, the modeler specifies only the parameter values, DEVS-Scheme does the rest.

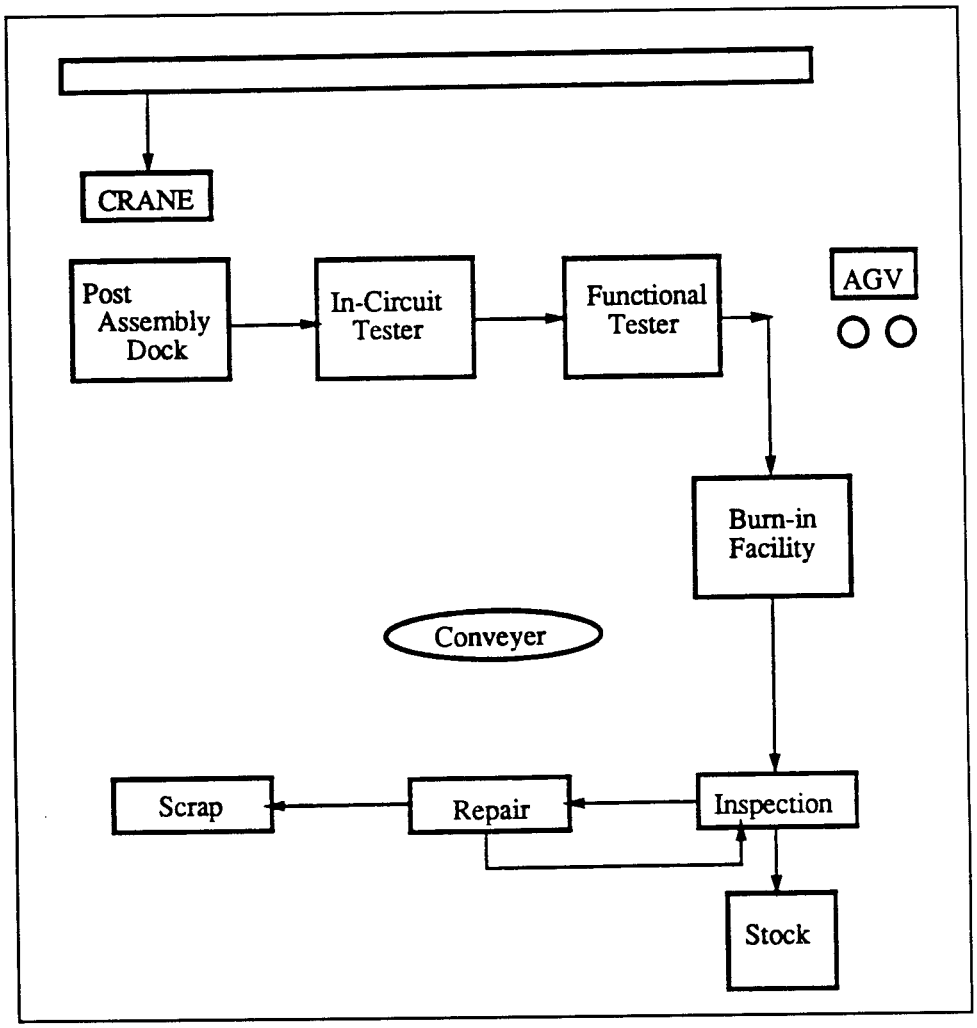


Figure 7. Architecture of Automatic Test Facility - Version 1

A new sub-class of *kernel-models, tree-models,* was developed for use in the testing methodology described here. As shown in Figure 10, the modeler can specify that the model architecture may be a network with tree topology containing

components linking test stati
structure of boards under tes
board. This mapping of board
test sites is a convenient repre
A mapping of such an ideal tre
necessary to fully study issues i
believe that many questions
addressed to the proposed tree

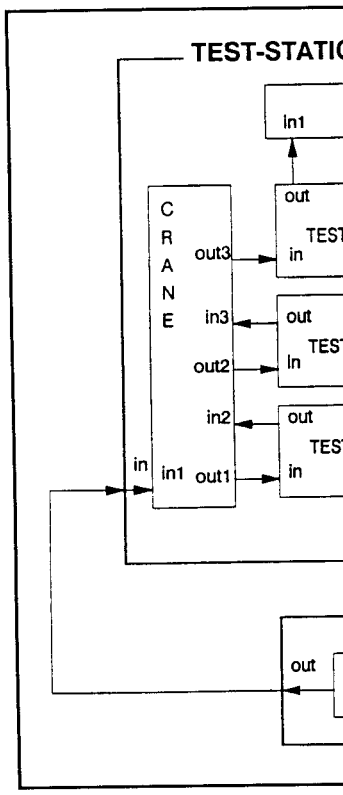
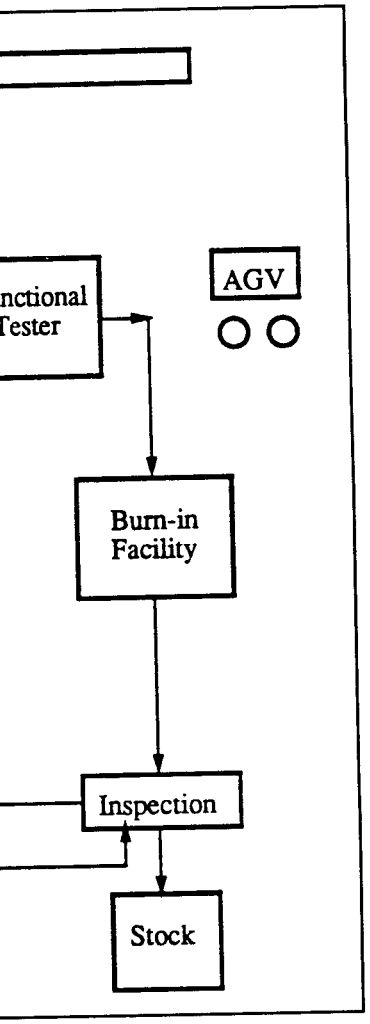


Figure 7a. Simulato

Routing is one such
specify various routing strate
modules in a board to be te
(Figure 11). However, to av
and decreasing total test ti
station would examine the pa
current work load of its neig
its next test. Other conside

stematic manner, we extended
Scheme, kernel-models and its
ting models having arbitrary
, the kernel, and coupled in a
termined by a parameterized
ls. Thus, the modeler specifies
t.



t Facility - Version 1

els, was developed for use in the
igure 10, the modeler can specify
with tree topology containing

components linking test stations. The tree structure represents the composition structure of boards under test. Thus each workstation represents a module of the board. This mapping of board components to a logically isomorphic configuration of test sites is a convenient representation to address the issues of the test methodology. A mapping of such an ideal tree network to a physically feasible network realization is necessary to fully study issues involving traveling distances and the like. However, we believe that many questions, of an organizational nature, can be meaningfully addressed to the proposed tree structured models.

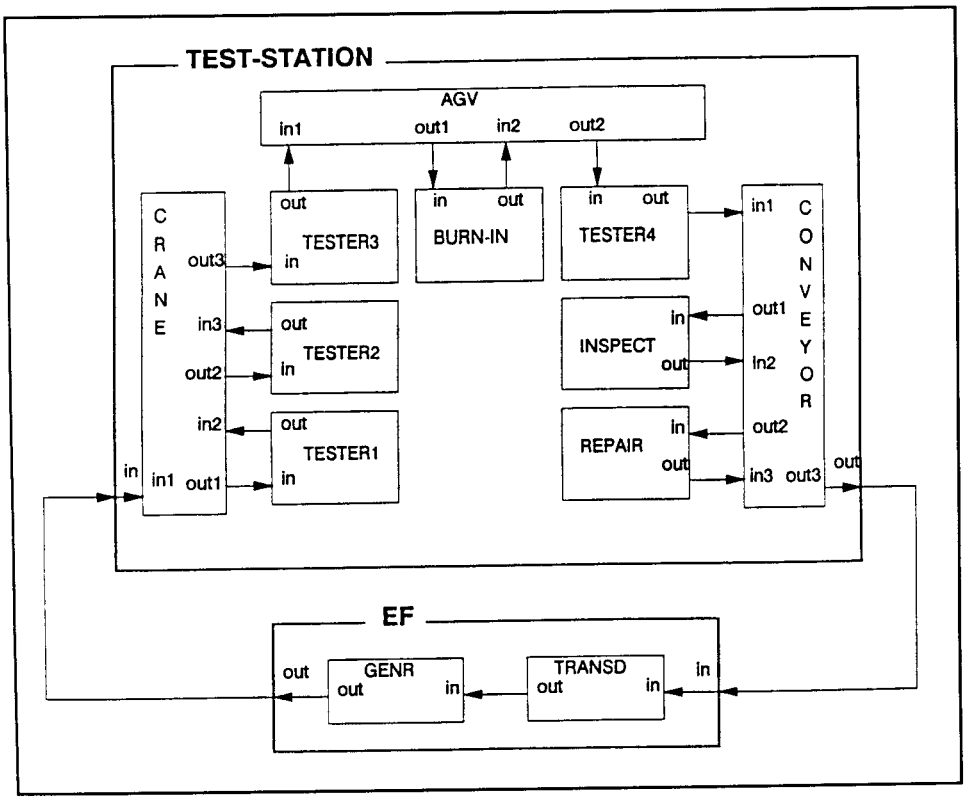


Figure 7a. Simulator Architecture for the Test Facility of Figure 7

Routing is one such issue. Rules can be provided to test stations to easily specify various routing strategies. For example, in a test strategy that requires all sub-modules in a board to be tested, we can readily specify a fixed traversal of the tree (Figure 11). However, to avoid bottlenecks, with the intent of increasing throughput and decreasing total test time, adaptive strategies can be defined. In these, a test station would examine the past history of the board it had just finished testing and the current work load of its neighbors in the tree, to decide where to send the board for its next test. Other considerations such as test level and test procedural constraints

(Section 3.3) can also be encoded in the test station model.

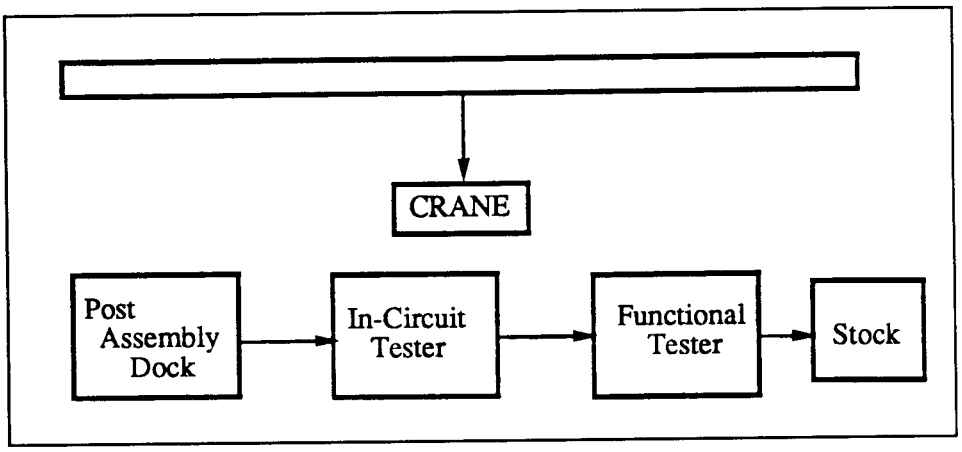


Figure 8. Architecture of the Automatic Test Facility - Version 2

3.7 Evaluation of Tester Performance

Various test strategy/architecture combinations can be generated by pruning as previously described and their performance measured through simulation in the prespecified experimental frames (Section 3.1). By comparison of such results, we can arrive at the test strategy and tester configuration that best satisfy our evaluation criteria with given tradeoff specifications.

4. SUMMARY

We have presented a conceptual basis for flexible and rapid modeling of an integrated test cell. The methodology is based on the formal concepts of Knowledge Based System Design and Simulation. The system under test is represented by the entity structure and fault locations are modeled in trees resulting from this representation. The tester configuration is generated from a generic set of test cells using a production-rule based approach. Alternative test strategies and models of a tester can be rapidly constructed and evaluated with respect to performance criteria defined by the test engineer. This approach is expected to result in improvements in test/rework time, first pass yield, and reduced scrap and work-in-process inventory.

REFERENCES

1. Agogino, A.M., et. al., 1989. AI/OR Computational Model for Integrating Qualitative and Quantitative Design Methods, *Proc. of NSF Engineering Design Research Conference,* Amherst, pp. 97-112.

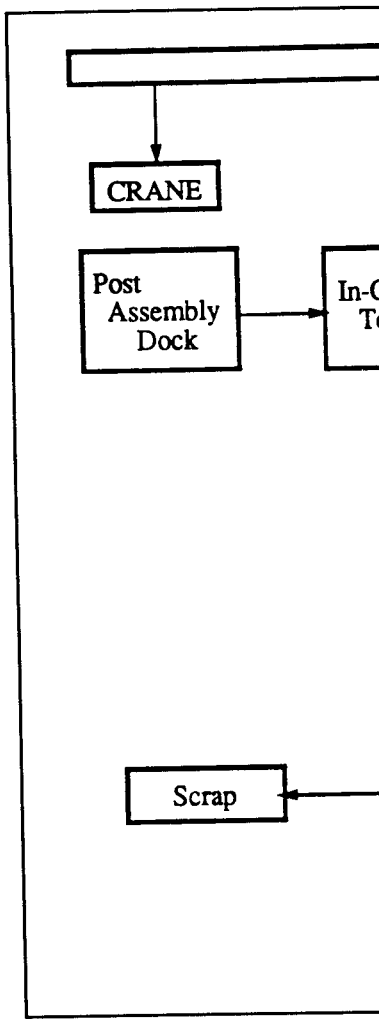


Figure 9. Architectur

2. Anderson, K.R., 1987. A Method
of the 1987 Winter Simulation C

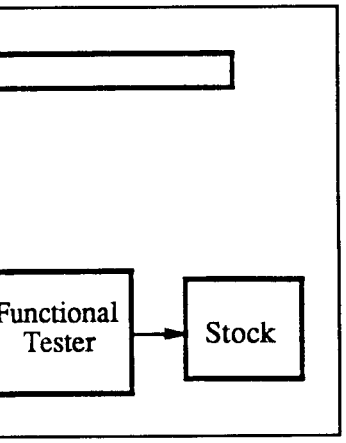
3. Anderson, K.R. and G.W. Diehl, 1
of the 1988 Winter Simulation C

4. Dixon, J.R. et. al., 1989. Comput
for Redesign, *Proc. of NSF Eng*

5. Elzas, M.S., 1986. The Appli
Representation in Modeling
Intelligence Era, (eds. Elzas, M

6. Feugate, R.J. and S.M. McIntyre,

del.



Facility - Version 2

rmance

ns can be generated by pruning
ıred through simulation in the
comparison of such results, we
that best satisfy our evaluation

xible and rapid modeling of an
formal concepts of Knowledge
nder test is represented by the
in trees resulting from this
from a generic set of test cells
test strategies and models of a
respect to performance criteria
ed to result in improvements in
d work-in-process inventory.

odel for Integrating Qualitative and
sign Research Conference, Amherst, pp.

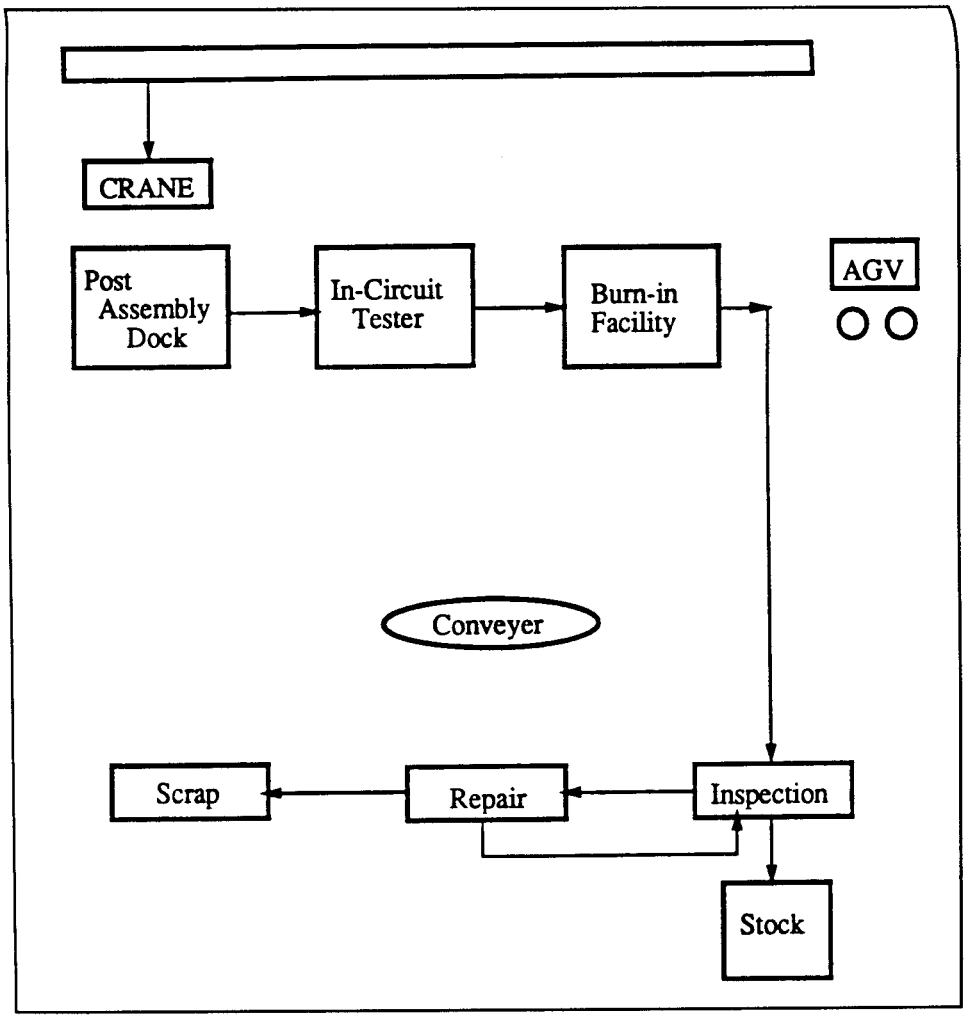


Figure 9. Architecture of the Automatic Test Facility - Version 3

2. Anderson, K.R., 1987. A Method for Planning Analysis and Design Simulation of CIM Systems, *Proc. of the 1987 Winter Simulation Conference,* Atlanta, GA, pp. 715-720.

3. Anderson, K.R. and G.W. Diehl, 1988. Rapid Prototyping: Implications for Business Planning, *Proc. of the 1988 Winter Simulation Conference,* San Diego, pp. 691-696.

4. Dixon, J.R. et. al., 1989. Computer-Based Models of Design Processes: The Evaluation of Designs for Redesign, *Proc. of NSF Engineering Design Research Conference,* Amherst, pp. 491-506.

5. Elzas, M.S., 1986. The Applicability of Artificial Intelligence Techniques to Knowledge Representation in Modeling and Simulation, in: *Modeling and Simulation in the Artificial Intelligence Era,* (eds. Elzas, M.S., Ören, T.I. and B.P. Zeigler), North-Holland, pp. 19-40.

6. Feugate, R.J. and S.M. McIntyre, 1988. *Introduction to VLSI Testing*, Prentice Hall.

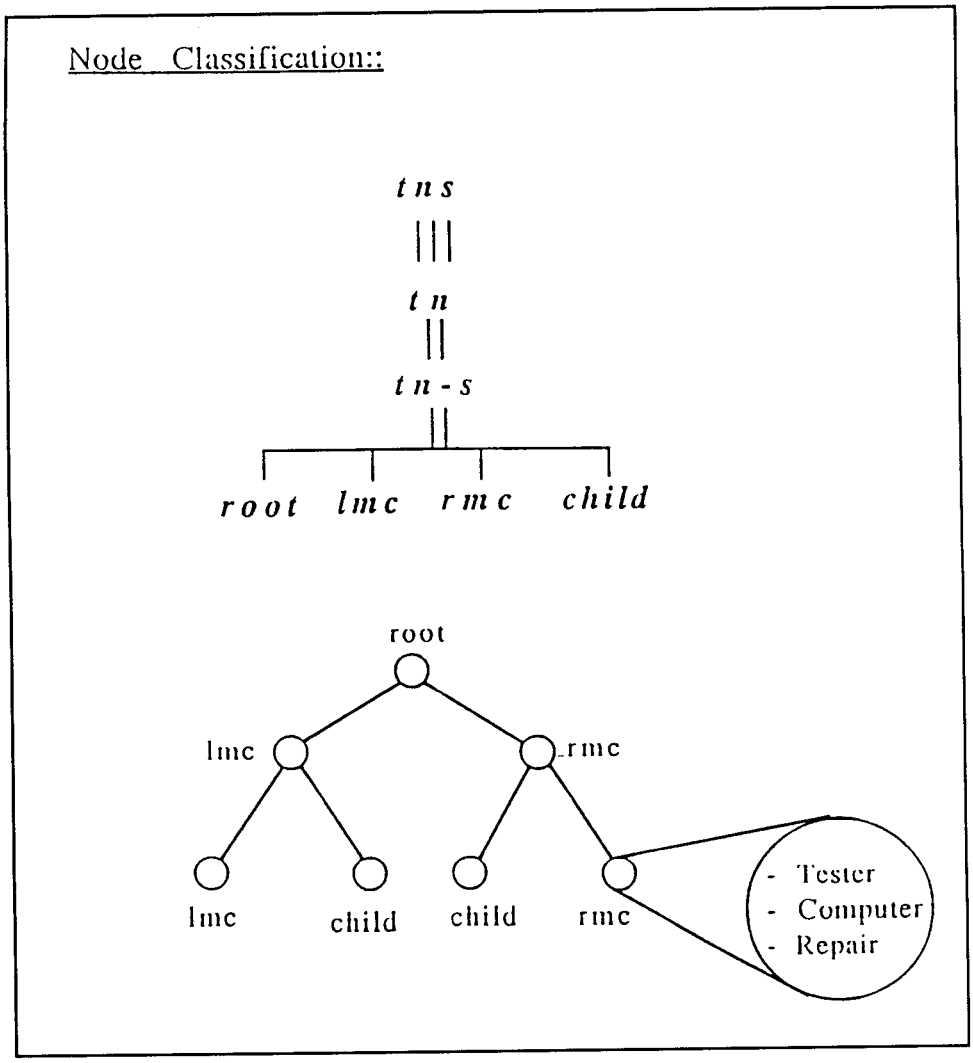


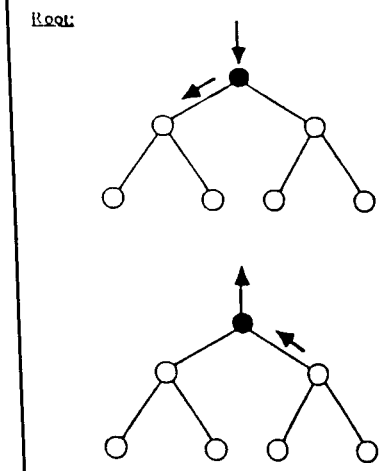
Figure 10. Tree Representation of Test Site

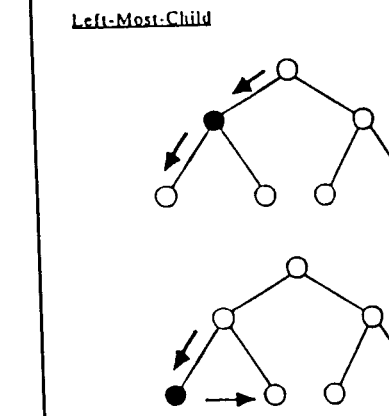
7. Gero, John S. et. al., 1989. An Approach to Knowledge-Based Creative Design, *Proc. of NSF Engineering Design Research Conference,* Amherst, pp. 333-346.

8. Gilmore, H.L. and H.C. Schwartz, 1986. *Integrated Product Testing and Evaluation: A Systems Approach to Improve Reliability and Quality,* Marcel Dekker.

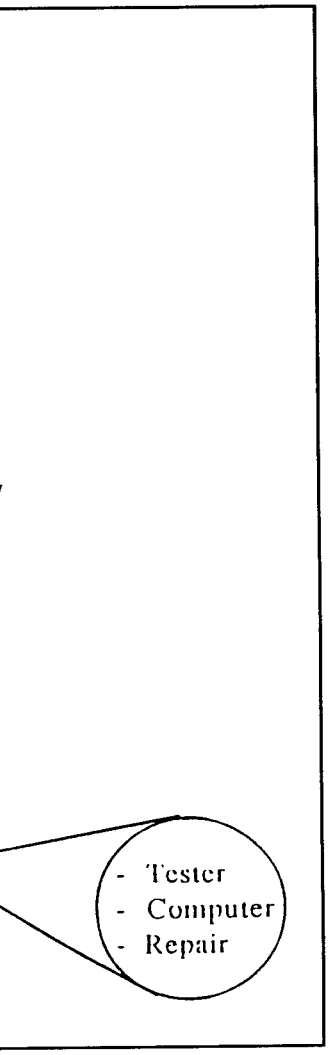
9. Gruber, T. and P. Cohen, 1987. Principles of Design for Knowledge Acquisition, *Proc. of IEEE 1987 Third Conference on Artificial Intelligence Application,* February, pp. 9-15.

10. Ho, Y., 1989. Editors Introduction, Special Issue on Dynamics of Discrete Event Systems, *Proceedings of the IEEE,* Vol. 77, No. 1.





Figure



Test Site

sed Creative Design, *Proc. of NSF*

Testing and Evaluation: A Systems

edge Acquisition, *Proc. of IEEE 1987*
ry, pp. 9-15.

namics of Discrete Event Systems,

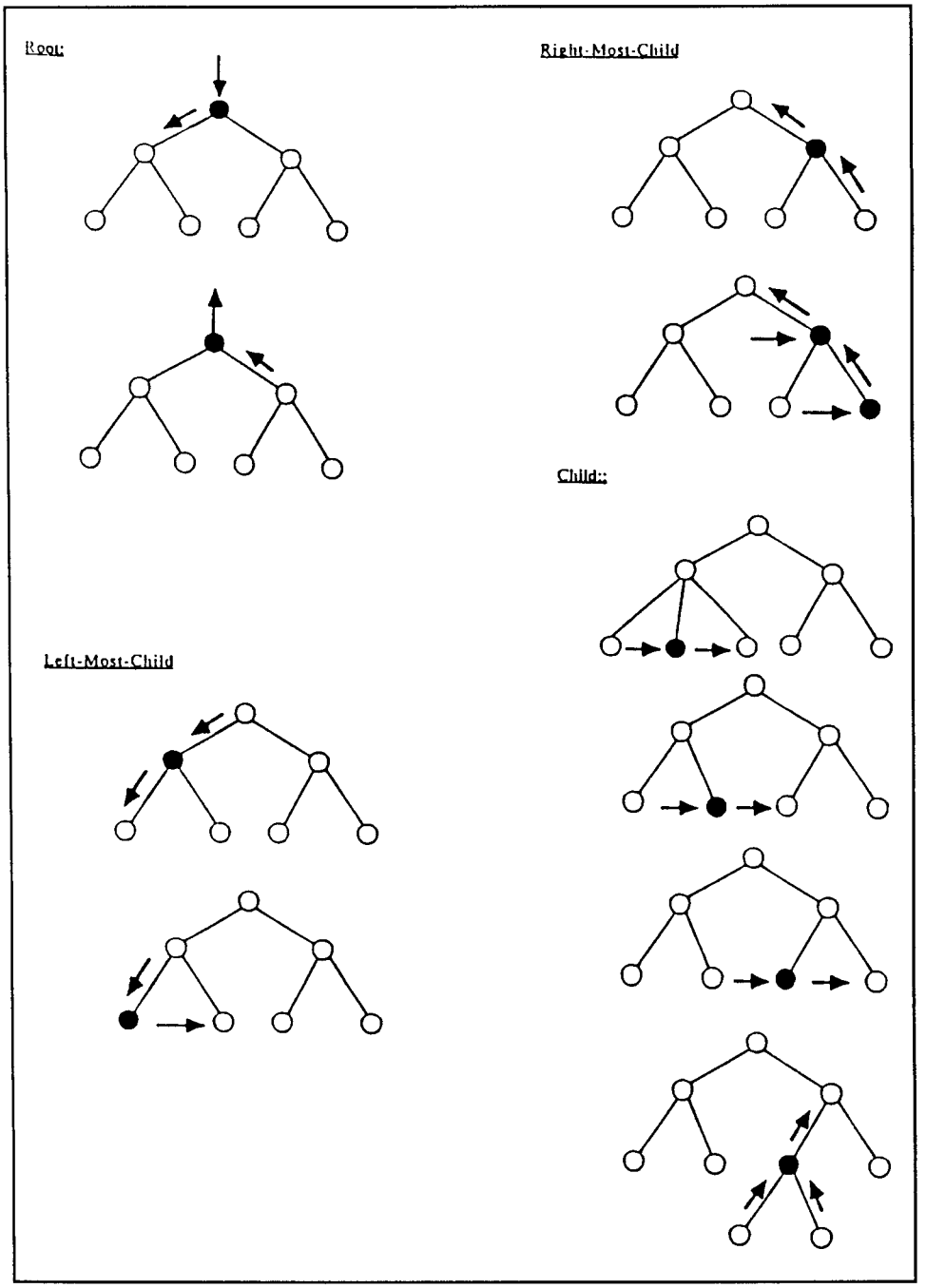


Figure 11. Diversity of Tree Traversals

11. Huang, Y.M., 1987. Building an Expert System Shell for Design Model Synthesis in Logic Programming, Master Thesis, University of Arizona, Tucson, Arizona.

12. *Introduction to Multi-Strategy Testing,* GenRad, Inc., Concord, MA, 1989.

13. Kear, F.W., 1987. *Printed Circuit Assembly Manufacturing,* Marcell Dekker.

14. Kim, Tag Gon, 1988. A Knowledge-Based Environment for Hierarchical Modeling and Simulation, Tech. Report AIS-7 (Ph.D. Thesis), Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ.

15. Kim, Tag Gon and B.P. Zeigler, 1990. The DEVS-Scheme Simulation and Modeling Environment, in Knowledge Based Simulation: Methodology and Application (eds: Paul A. Fishwick and Richard B. Modjeski), Springer Verlag Inc. (to appear).

16. Mesarovic, M.D. and Y. Takahara, 1975. *General Systems Theory: Mathematical Foundations,* Academic Press.

17. Nilsson, N.J., 1980. *Principles of Artificial Intelligence,* Tioga, Palo Alto, CA.

18. Ören, T.I., 1984. GEST - A Modeling and Simulation Language Based on System Theoretic Concepts in: *Simulation and Model-Based Methodologies: An Integrative View,* (eds: Ören, T.I., Zeigler, B.P. and M.S. Elzas), North Holland, pp. 3-40.

19. Rozenblit, J.W., 1985. A Conceptual Basis for Integrated, Model-Based System Design, Ph.D. Thesis, Department of Computer Science, Wayne State University, Detroit, Michigan.

20. Rozenblit, J.W. and B.P. Zeigler, 1986. Entity-Based Structures for Model and Experimental Frame Construction, in: *Modeling and Simulation in Artificial Intelligence Era,* in: (eds. Elzas, M.S., Ören, T.I. and B.P. Zeigler), North-Holland, pp. 78-100.

21. Rozenblit, J.W. and Zeigler, B.P., 1988. Design and Modeling Concepts in: *International Encyclopedia of Robotics, Applications and Automation,* (ed. Dorf, R.) John Wiley and Sons, New York, pp. 308-322.

22. Rozenblit, J.W. and Y. Huang, 1987. Constraint-Driven Generation of Model Structures, in *Proc. of 1987 Winter Simulation Conf.,* Atlanta, GA, pp. 604-611.

23. Rozenblit, J.W., Hu, J. and Y.M. Huang, 1989. An Integrated, Entity-Based Knowledge Representation Scheme for System Design, *Proc. of NSF Engineering Design Research Conference,* Amherst, pp. 393-408.

24. Rozenblit, J.W., Hu, J.F., Kim, T.G. and B.P. Zeigler, 1990. Knowledge-Based Design and Simulation Environment (KBDSE): Foundational Concepts and Implementation, *Journal of Operational Research Society,* Vol. 41, No. 6, pp. 475-489.

25. Sage, A.P., 1977. *Methodology for Large Scale Systems,* Mc-Graw Hill, New York.

26. Winston, P.H., 1977. *Artificial Intelligence, 2nd Ed.,* Addison-Wesley, Massachusetts.

27. Wymore, A.W., 1967. *A Mathematical Theory of Systems Engineering: The Elements,* John Wiley.

28. Zadeh, L.A. and C.A. Desoer, 1963. *Linear Systems Theory, The State Space Approach,* McGraw Hill.

29. Zeigler, B.P., 1984. *Multifacetted Modeling and Discrete Event Simulation,* Academic Press.

30. Zeigler, B.P., 1984. System-Theoretic Representation of Simulation Models, *IIE Trans.,* 16, 10-27.

31. Zeigler, B.P., 1987. Hierar
Environment, *Simulation,* V

32. Zeigler, B.P., 1990. *Object-Ori*
and Endomorphic Systems, A

33. Zeigler, B.P. and Q. Zhang, 19
Systems: Algorithm, Analy
press).

34. Yang, J. and J.W. Rozenblit, 1
International Conference o
Computer Press, pp. 136-141

esign Model Synthesis in Logic
zona.

989.

Dekker.

rchical Modeling and Simulation,
al and Computer Engineering,

on and Modeling Environment, in
ion (eds: Paul A. Fishwick and

ory: *Mathematical Foundations,*

lto, CA.

sed on System Theoretic Concepts
egrative View, (eds: Ören, T.I.,

ased System Design, Ph.D. Thesis,
roit, Michigan.

r Model and Experimental Frame
igence Era, in: (eds. Elzas, M.S.,

ling Concepts in: *International*
orf, R.) John Wiley and Sons, New

n of Model Structures, in *Proc. of*

ated, Entity-Based Knowledge
eering Design Research Conference,

edge-Based Design and Simulation
mentation, *Journal of Operational*

ll, New York.

y, Massachusetts.

g: *The Elements,* John Wiley.

te Space Approach, McGraw Hill.

lation, Academic Press.

n Models, *IIE Trans.,* 16, 10-27.

31. Zeigler, B.P., 1987. Hierarchical, Modular Discrete-Event Models in an Object-Oriented Environment, *Simulation,* Vol. 50, No. 5, pp. 219-230.

32. Zeigler, B.P., 1990. *Object-Oriented Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems,* Academic Press.

33. Zeigler, B.P. and Q. Zhang, 1990. Mapping Hierarchical Discrete Event Models to Multiprocessor Systems: Algorithm, Analysis and Simulation, *Journal on Parallel and Distributed Computers* (in press).

34. Yang, J. and J.W. Rozenblit, 1990. Case Studies of Design Methodologies: A Survey, *Proc. of the International Conference on AI, Simulation and Planning in High Autonomy Systems,* IEEE Computer Press, pp. 136-141.